Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jošt Rovtar

Implementacija poslovnih pravil z uporabo sistema za upravljanje s poslovnimi pravili

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana, 2017

Zahvaljujem se mentorju prof. dr. Matjažu Juriču za usmerjanje pri izbiri diplomske teme ter pomoč pri njeni izvedbi. Zahvaljujem se mu tudi za možnost sodelovanja na projektu Po kreativni poti do praktičnega znanja v Laborarotiju za integracijo informacijskih sistemov.

Zahvaljujem se tudi staršem in ženi za vso pomoč, razumevanje in spodbujanje v času svojega študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ABRD	Agile Business Rule Development	agilen pristop razvoja poslovnih pravil
BLIP	Business Logic Integration Platform	platforma za integracijo poslovne logike
BPEL	Business Process Execution Language	programski jezik za izvajanje poslovnih procesov
BPMN 2.0	Business Process Model and Notation	notacija za modeliranje in izvajanje poslovnih procesov
BPMS	Business Process Management System	sistem za upravljanje poslovnih procesov
BRMS	Business Rule Management System	sistem za upravljanje s poslovnimi pravili
DSL	Domain Specific Language	naravni jezik domene
ER	Entity – Relationship	entitetno-relacijski
GPL	General Public License	licenca za brazplačno uporabo programskega produkta
JMS	Java Message Service	sporočilna storitev java
KIE	Knowledge Is Everything	znanje je vse
REST	Representational State Transfer	predstavitveni prenos stanja
SOA	Service Oriented Architecture	storitveno orientirana arhitektura

SOAP	Simple Object Access Protocol	protokol za izmenjavo enostavnih objektov
SQL	Structured Query Language	strukturirani povpraševalni jezik za delo s podatkovnimi zbirkami
URL	Uniform Resource Locator	enotni lokator vira
VHDL	VHSIC Hardware Description Language	VHSIC jezik za opisovanje delovanja integriranih vezij
VHSIC	Very High Speed Integrated Circuit	zelo hitra integrirana vezja
XML	Extensible Markup Language	razširljivi označevalni jezik

Povzetek

Naslov: Implementacija poslovnih pravil z uporabo sistema za upravljanje s poslovnimi pravili

Avtor: Jošt Rovtar

V diplomski nalogi so predstavljeni sistemi za upravljanje s poslovnimi pravili (sistemi BRMS), njihova zgodovina, zgradba ter prednosti in slabosti uvedbe teh sistemov. Predstavljeni so možni načini razvoja programske opreme z njihovo uporabo ter dve formalni metodologiji. Predstavljena so tudi poslovna pravila. Navedenih je nekaj njihovih definicij in klasifikacij, predstavljeni pa so tudi najpogostejši načini zapisa. Ker podjetja običajno nimajo zbranih vseh poslovnih pravil, je opisana tudi metoda za pridobitev vseh poslovnih pravil, ki jih podjetje uporablja. Sledi predstavitev nekaj pogosto uporabljenih produktov BRMS. Na koncu je opisana implementacija poslovnih pravil, ki smo jo izvedli s pomočjo produktov družine Drools.

Ključne besede: BRMS, poslovna pravila, Drools.

Abstract

Title: Implementation of business rules using business rule management system

Author: Jošt Rovtar

In this thesis we present Business Rules Management Systems (BRMS) and their history, general structure, advantages and disadvantages. We describe different methods of software development with the help of these systems and two principal methodologies. Business rules are also presented along with some basic definitions, classifications and most frequently used record formats. Generally, companies do not have all their business rules assembled, therefore, methods for gathering all business rules are described. We also introduced some commonly used commercial and open source BRMS products. At the end we provide a description of the process of implementation of business rules, which was done by using Drools technology.

Keywords: BRMS, business rules, Drools.

Poglavje 1

Uvod

Podjetja se danes srečujejo z veliko izzivi. Pritisk konkurence in želja po zadovoljitvi strank jih silita, da neprestano razmišljajo o optimizaciji poslovnih procesov in hitrem prilagajanju poslovnih odločitev. Sposobnost hitrega prilagajanja spremembam v poslovnem svetu je v današnjem času zelo pomemben faktor pri ohranjanju konkurenčnosti in uspešnosti podjetja.

Za velika podjetja je značilno, da za podporo poslovanja uporabljajo več različnih aplikacij. Komponente teh aplikacij, ki implementirajo omejitve, principe in pravila poslovanja podjetja, se pod skupnim imenom imenujejo poslovna logika. Ta je tipično raztresena po različnih aplikacijah, kar lahko privede do več različnih implementacij istih pravil. Lahko se zgodi tudi, da poslovna logika vsebuje pravila, ki poslovnim uporabnikom niso znana, ali obratno, da poslovni uporabniki poznajo in uporabljajo določena pravila, ki niso implementirana v poslovni logiki. Poleg tega poslovna logika ni dostopna poslovnim uporabnikom, temveč jo lahko spreminjajo le razvijalci programske opreme. To se izkaže kot problem predvsem takrat, ko je glede na potrebe poslovanja potrebno spremeniti poslovno logiko v vseh podpornih aplikacijah. Poslovni uporabniki morajo pripraviti opis sprememb poslovne logike, razvijalci programske opreme pa glede na spremembe prilagoditi podporno programsko opremo. Opisan postopek vzame precej časa in truda, pojavi pa se tudi možnost večje nekonsistence poslovne logike, če razvijalec ne spremeni poslovnih pravil na vseh mestih.

Zaradi naštetih slabosti se v zadnjem času vse bolj uveljavlja razvoj informacijskih sistemov po principu poslovnih pravil (angl. Business Rule Approach). Ta

zapoveduje, da morajo biti vsa poslovna pravila zbrana in predstavljena v posebnem centralnem sistemu, ki skrbi za celotno upravljanje s poslovnimi pravili. Take sisteme imenujemo sistemi za upravljanje s poslovnimi pravil oz. sistemi BRMS (angl. Business Rule Management System). Njihova uporaba ne vpliva le na razvoj aplikacij in njihove razvijalce, temveč tudi na poslovne uporabnike oz. poslovne analitike, ki so odgovorni za posodabljanje in dodajanje novih poslovnih pravil v sistem.

Cilj diplomske naloge je bil preučiti sisteme BRMS in poslovna pravila, podati spremembe pri razvoju programske opreme z uporabo omenjenih sistemov ter prikazati njihovo uporabo na praktičnem primeru. V drugem poglavju so predstavljeni sistemi BRMS. Predstavljeno je njihovo zgodovinsko ozadje, njihove glavne značilnosti ter spremembe pri razvoju programske opreme ob uporabi omenjenih sistemov. V tretjem poglavju so opisana poslovna pravila, kjer je naštetih nekaj njihovih definicij, klasifikacij in najbolj pogosti načini njihovega zapisa. Opisana je tudi metoda za pridobivanje poslovnih pravil iz zapuščinskih sistemov. V četrtem poglavju so predstavljani nekateri produkti BRMS. V zadnjem poglavju je predstavljena izdelava praktičnega primera. Ta obsega postavitev sistema BRMS, implementacijo poslovnih pravil spletne trgovine, njihovo postavitev v izvajalno okolje ter implementacijo testov posameznih sklopov poslovnih pravil.

Poglavje 2

Sistemi za upravljanje s poslovnimi pravili

Ker so se sistemi BRMS razvili iz sorodnih produktov, bomo v tem poglavju najprej predstavili njihovo zgodovinsko ozadje. V nadaljevanju bomo opisali njihovo splošno zgradbo, podali njihove prednosti in slabosti ter predstavili dve formalni metodologiji razvoja programske opreme s pomočjo teh sistemov. Na koncu poglavja bomo predstavili še priporočila za umestitev sistemov BRMS v arhitekturo informacijskega sistema.

2.1 Zgodovinsko ozadje

Moderni sistemi BRMS so se razvili iz prvih ekspertnih sistemov (angl. Expert Systems) in kasnejših sistemov za upravljanje z znanjem (angl. Knowledge Management Systems).

Eden prvih najbolj znanih ekspertnih sistemov je bil MYCIN [20, 27]. Omogočal je identifikacijo določenih nevarnih bakterij ter na podlagi tega predlagal primeren antibiotik in odmerek glede na podano bolnikovo težo. Ime omenjenega ekspertnega sistema izvira iz same funkcionalnosti, saj se veliko antibiotikov v angleškem jeziku konča na -mycin. Razvil ga je Edward Shortliffe leta 1975 na univerzi v Stanfordu. Testiranje omenjenega ekspertnega sistema MYCIN je pokazalo, da je sistem predlagal pravilno terapijo v približno 70% primerih. Zaradi pravnih in

etičnih težav ni bil nikoli sprejet v praktično uporabo. Ekspertni sistem MYCIN ni bil v nobenem pogledu sistem BRMS, saj je vseboval pravila, ki so bila zakodirana v izvorno kodo in omejena zgolj na medicinsko domeno. Od ostalih podobnih sistemov se je bistveno razlikoval predvsem po poslovnih pravilih, ki so bila ločena od kontrolne logike. Poleg tega pa so bila pravila napisana deklarativno in ne imperativno. To sta glavni značilnosti, ki sta bistveni tudi za moderne sisteme BRMS. Ob spoznanju odkritega potenciala omenjenega sistema so mu odstranili poslovna pravila, ga prilagodili za splošno uporabo ter preimenovali v EMYCIN (angl. Empty MYCIN).

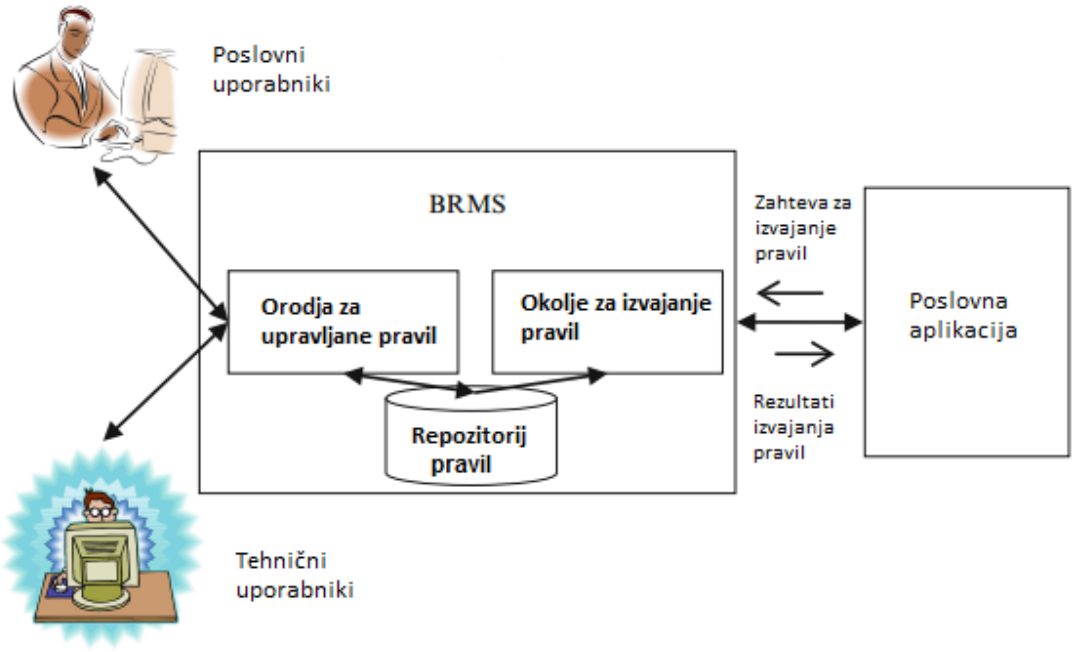
Ob koncu osemdesetih let prejšnjega stoletja so se raziskovalci in razvijalci začeli zavedati vpliva ekspertnih in inteligentnih sistemov na nadaljnji razvoj programske opreme. Sistemi BRMS so ustvarili novo vejo razvoja programske opreme in kmalu so se začeli pojavljati prvi samostojni programski produkti, namenjeni shranjevanju in izvajanju poslovnih pravil. Med njimi sta bila tudi ILOG SA (1985) in Blaze Advisor (1988), predhodnika še danes zelo popularnih produktov IBM ODM ter FICO Blaze Advisor. Prvi sistemi BRMS so bili zasnovani predvsem na podatkovnih zbirkah [28]. Nekateri sistemi so imeli poslovna pravila implementirana kot shranjene procedure (angl. Stored Procedures), drugi kot zapise v tabeli, najbolj napredni pa kot prožilce (angl. Triggers).

2.2 Splošna zgradba sistema BRMS

Sistem BRMS mora celostno podpirati metodologijo razvoja programske opreme po principu poslovnih pravil. Omogočati mora centralno shranjevanje poslovnih pravil ter njihovo urejanje in izvajanje. Zato so moderni sistemi BRMS sestavljeni iz naslednjih komponent:

- Repozitorija, v katerem so zbrana vsa poslovna pravila;
- Orodij, ki poslovnim in tehničnim uporabnikom omogočajo upravljanje s poslovnimi pravili v repozitoriju;
- Izvajalnega okolja, ki skrbi za izvršitev poslovnih pravil.

Slika 2.1 prikazuje zgradbo sistema BRMS znotraj operativnega okolja.



Slika 2.1: Splošna zgradba sistema BRMS [26].

Ključna prednost vpeljave metodologije razvoja programske opreme po principu poslovnih pravil je, da so za upravljanje poslovnih pravil zadolženi poslovni in ne tehnični uporabniki. Zato je zelo pomembno, da so orodja za upravljanje s poslovnimi pravili prilagojena njim.

Nekateri sistemi BRMS omogočajo neodvisno uporabo posameznih komponent, kar je zelo uporabno pri manjših ali prototipnih projektih.

2.3 Prednosti sistema BRMS

Uspešna integracija sistema BRMS v informacijski sistem podjetja in njegova pravilna uporaba privede do določenih poslovnih in tehničnih prednosti. Nekaj smo jih že omenili, spodaj pa so naštete vse bistvene [28, 37]:

- **Deklarativen zapis pravil.** Ko zapisujemo poslovna pravila, definiramo samo, *kaj* naj se zgodi (ob določenih pogojih), ne pa tudi *kako* naj pridemo

do rezultata. To pomeni, da se nam ni potrebno več ukvarjati z izvajalno logiko in se lahko posvetimo le poslovnim pravilom. Podoben primer deklarativnega jezika je poizvedovalni jezik SQL. V poizvedbi SQL navedemo le vsebinske lastnosti njenih rezultatov, ne pa tudi kako naj iskanje teh rezultatov poteka.

- **Večja berljivost.** Vsak sistem BRMS ima definirano sintakso, s katero so poslovna pravila zapisana. Nova sintaksa sicer predstavlja dodatno učenje za nove uporabnike, obenem pa omogoča enoten zapis vseh pravil, kar privede do bolj berljivih in hitreje razumljivih poslovnih pravil.

- **Lažje vzdrževanje.** Posledica večje berljivosti je tudi lažje vzdrževanje. Zaradi enotne sintakse pravila lažje posodabljamo, obenem pa hitreje odkrivamo in odpravljamo morebitne nastale napake.

- **Ponovna uporaba.** Ker so pravila zbrana na enem mestu, je uporaba v več aplikacijah preprosta. S tem zagotovimo tudi konsistentnost, saj so vsa pravila definirana enkrat in na enem mestu.

- **Zmogljivost.** Večina izvajalnih okolij za poslovna pravila (angl. Rule Engine) temelji na naprednih algoritmih (npr. RETE, LEAPS), ki določajo, katera pravila se bodo izvedla na podlagi vhodnih podatkov. Ti algoritmi so matematično dokazano hitrejši in v primerjavi s klasičnim zapisom v programski kodi omogočajo večjo skalabilnost pri velikem številu pravil.

- **Ločen življenjski cikel.** Ker se poslovna logika spreminja hitreje kot ostali deli aplikacij, je pomembno, da se njen razvoj odvija v ločeni aktivnosti. Le tako lahko omogočimo dovolj hitro uveljavitev sprememb poslovnih pravil ter posledično pospešimo poravnavo med potrebami poslovanja ter tehnično podporo.

- **Hitrejši razvoj.** Razvijalcem se ni potrebno ukvarjati z zapisom poslovne logike, kar posledično pripelje do hitrejšega razvoja novih aplikacij.

Poleg zgoraj naštetih prednosti imajo sistemi BRMS tudi slabosti [25, 28]. V nadaljevanju so našteta dejstva, ki se jih moramo zavedati in upoštevati pri uvedbi sistema BRMS.

- **Izobraževanje razvijalcev.** Razvijalci, ki niso primerno usposobljeni ne morejo implementirati učinkovitih produktov. Za implementacijo poslovnih pravil bodo potrebovali več časa, večja pa je tudi verjetnost, da se poslovna pravila ne bodo izvajala optimalno. Razvijalci se morajo prilagoditi na drugačen način razmišljanja ter uporabiti deklarativni način zapisa poslovnih pravil. To velja tako za razvijalce programske opreme kot tudi za poznavalce poslovne domene, ki bodo zadolženi za pisanje poslovnih pravil.

- **Odpravljanje napak.** Okolja za izvajanje poslovnih pravil temeljijo na kompleksnih algoritmih. Nepoznavanje delovanja teh algoritmov oz. delovanja izvajalnega okolja prinaša veliko težav pri odkrivanju in odpravljanju napak. S tem namenom so bila razvita tudi orodja za razhroščevanje poslovnih pravil in so vključena v praktično vsak modern sistem BRMS. Ta orodja so nam sicer v veliko pomoč, ne morejo pa nadomestiti potrebnega znanja o delovanju izvajalnih okolij.

- **Delovni pomnilnik.** Naslednja slabost izvajalnih okolij poslovnih pravil je poraba delavnega pomnilnika. Algoritmi za potrebe izračunov gradijo določene podatkovne strukture, ter jih začasno shranjujejo v delovni pomnilnik z namenom kasnejše uporabe. Ker je cena pomnilnikov danes relativno nizka, ta problem ni tako velik, a je pomembno, da se ga zavedamo.

- **Enotna točka odpovedi** (angl. Single Point Of Failure). Centralizacija poslovnih pravil posledično prinaša veliko odvisnost aplikacij od sistema BRMS. Enotna točka upravljanja in dostopa pomeni tudi enotno točko odpovedi.

- **Odvisnost od izbranega produkta.** Vsak produkt BRMS ima specifične lastnosti. Večinoma pa se vsi razlikujejo po sintaksi zapisa poslovnih pravil. V primeru, da trenutni ponudnik preneha z razvojem produkta in zanj ukine podporo, je potrebno izbrati novega, ga uvesti in prepisati vsa obstoječa pravila, kar zahteva velik časovni in finančni vložek.

Glede na lastnosti sistemov BRMS ter dejstev, ki se jih je potrebno zavedati pri uvedbi, so spodaj zapisani primeri, ko uvedba sistema BRMS ni primerna:

- Če je projekt majhen in ne vsebuje veliko (manj kot 50) poslovnih pravil.

- Če je poslovna logika relativno preprosta, dobro definirana, se ne spreminja pogosto in ni potrebe po hitri uveljavitvi njenih sprememb.
- Če je zelo pomembna zmogljivost oz. hitrost sistema. Ne glede na zmogljivosti izvajalnih okolij poslovnih pravil je njihovo delovanje še vedno prepočasno za uporabo v določenih postopkih oz. algoritmih (npr. algoritem za obdelovanje videa).
- Če nimamo časa ali denarja za izobraževanje zaposlenih.

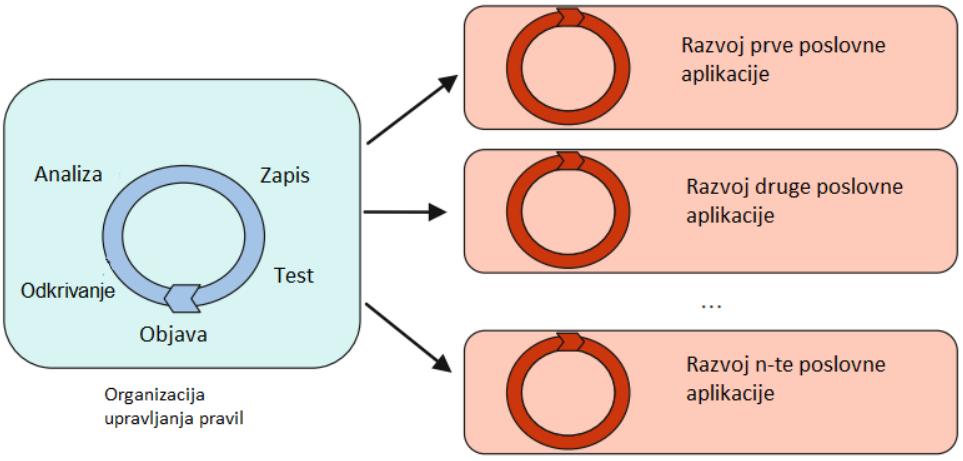
2.4 Razvoj programske opreme z uporabo sistema BRMS

Uvedba sistema BRMS prinese v razvojni cikel programske opreme določene spremembe. V splošnem imamo na voljo dve metodologiji:

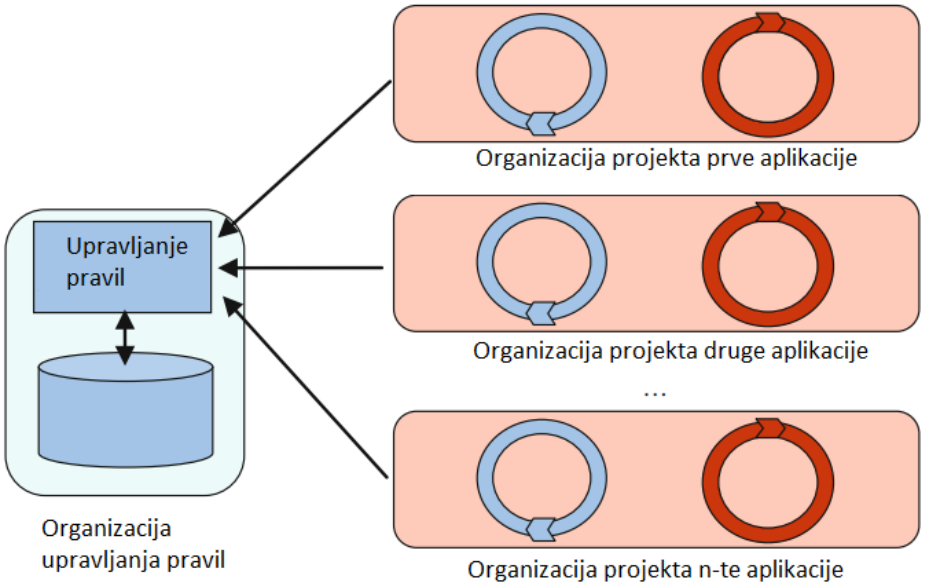
- Asinhrono: poslovna pravila lahko razvijamo v ločeni aktivnosti (kot ločen projekt). Imajo svoj življenjski cikel, ki je neodvisen od ostalih poslovnih aplikacij.
- Sinhrono: poslovna pravila lahko razvijemo kot stranski produkt določene poslovne aplikacije. V tem primeru bodo pravila razvita inkrementalno in vedno znotraj konteksta poslovne aplikacije. Ne glede na razvoj pa bodo shranjena in upravljana v ločenem repozitoriju.

Asinhrona metodologija je primerna za večja podjetja z ločeno skupino razvijalcev, ki so zadolženi za ustvarjanje in upravljanje vseh poslovnih pravil znotraj podjetja. Ta scenarij prikazuje slika 2.2. Zahteva veliko začetno investicijo v obliki denarnih in človeških virov. Prednosti te metodologije so boljša preglednost, večja koherentnost in konsistentnost poslovnih pravil.

Sinhrona metodologija za razliko od predhodne ne zahteva velikega začetnega vložka. Poslovna pravila se razvijajo vzporedno z razvojem aplikacije. Ta scenarij prikazuje slika 2.3. Slabost te metodologije je, da se lahko pravila na različnih projektih podvajajo. V najslabšem primeru so lahko ista pravila na različnih projektih implementirana različno oz. napačno.



Slika 2.2: Asinhron razvoj poslovnih pravil [26].



Slika 2.3: Sinhron razvoj poslovnih pravil [26].

V praksi podjetja uporabljajo vmesno metodologijo. Prvih nekaj projektov izpeljejo po principu, ki ga opisuje sinhrona metodologija. Velika verjetnost je, da bodo pri razvoju poslovnih pravil za potrebe druge aplikacije sodelovali isti

ljudje in si tako pridobili še dodatne izkušnje. Po nekaj projektih bodo razvili dovolj širok pogled na poslovna pravila podjetja. S tem bodo lahko oblikovali skupino, ki bo odgovorna za upravljanje vseh poslovnih pravil v podjetju in bo razvijala poslovna pravila po principu asinhrone metodologije. Oblikovana skupina je tipično sestavljena iz sledečih poslovnih in tehničnih vlog [8]:

- **Projektni vodja** (angl. Project Lead), ki definira izvedbeni plan.
- **Arhitekt poslovnih sistemov** (angl. Enterprise Architect), ki definira tehnično arhitekturo in integracijo sistema BRMS v informacijski sistem. Zadolžen je tudi za preverjanje ponovne uporabe implementiranih odločitvenih komponent in storitev v celotnem informacijskem sistemu. Na ta način dosežemo najboljši možen izkoristek sistema BRMS.
- **Poslovni analitik** (angl. Business Policy Analyst), ki organizira poslovna pravila v logične kategorije in definira strukturo poslovnih pravil.
- **Razvijalec poslovnih pravil** (angl. Rule Developer) je odgovoren za implementacijo poslovnih pravil.
- **Poznavalec poslovne domene** (angl. Internal Business Expert) je odgovoren za vzdrževanje poslovnih pravil iz področja njegove domene ter določanje ciljev aplikacij, ki uporabljajo poslovna pravila.

V nekaterih primerih je dobro vključiti tudi poznavalca delovne metodologije, ki pomaga skupini pri izpolnjevanju ciljev metodologije in združuje skupino v bolj učinkovito celoto. V primerih modernizacije informacijskega sistema iz zastarelih tehnologij je za učinkovito pridobitev poslovnih pravil potrebno vključiti tudi poznavalce te zastarele tehnologije.

Zaradi potreb po razvoju programske opreme, ki temeljijo na principu poslovnih pravil, so se v preteklosti razvile formalne metodologije razvoja. Najbolj znani metodologiji STEP [36] in ABRD [26] sta opisani v nadaljevanju.

2.4.1 Metodologija STEP

Metodologijo STEP je leta 2001 razvila Barbara von Halle. Predstavlja napredno sinhrono metodologijo razvoja poslovnih aplikacij na osnovi poslovnih pravil, ki

opisuje zgradbo same aplikacije in komponento s poslovnimi pravili. Temelji na štirih principih, po katerih je tudi dobila ime: **ločevanje** (angl. Separate), **sledenje** (angl. Trace), **zunanja hramba** (angl. Externalize) in **umeščanje** (angl. Position).

Princip ločevanja v tem primeru pomeni, da ločimo poslovna pravila od ostalih zahtev za implementacijo sistema. S tem postopkom dosežemo želen nivo ponovne uporabe poslovnih pravil. Če poslovna pravila obravnavamo kot ločeno entiteto, nam to omogoča neodvisen razvoj in upravljanje.

Sledenje zapoveduje, da za vsako pravilo hranimo dve ločeni povezavi, obenem pa nam omogoča ovrednotenje vpliva spremembe posameznega poslovnega pravila. Prva povezava vodi do izvira pravila, ki opisuje, zakaj je pravilo nastalo in kaj so bili cilji. Druga povezava vodi do implementacije poslovnega pravila.

Princip zunanje hrambe določa, da so pravila zbrana v nekem zunanjem sistemu. Zapisana morajo biti v obliki, ki je razumljiva netehničnim oz. poslovnim uporabnikom. S tem zagotovimo, da poslovni uporabniki poslovna pravila lažje najdejo, jih razumejo ter glede na potrebe dodajajo in spreminjajo.

Princip umeščanja definira možnost spreminjanja poslovnih pravil skladno s spremembami poslovanja podjetja. Vse spremembe morajo biti enostavne narave in izvedene hitro.

2.4.2 Metodologija ABRD

Metodologija ABRD (angl. Agile Businss Rule Development) je sinhrona metodologija, ki jo je leta 2003 razvila organizacija ILOG [26]. Vključuje vse akterje, aktivnosti in najboljše prakse za razvoj programske opreme. Metodologija ABRD zagotavlja inkrementalen in iterativen postopek razvoja programske opreme, ki vključuje nove koncepte za vključitev sistemov BRMS in BPMS (angl. Business Process Management System) v poslovne aplikacije. Omogoča boljše sovpadanje in prilagajanja tehnične podpore poslovanju. Zelo poudarja agilnost, kar jo najbolj razlikuje od ostalih metodologij. Drži se smernic in vrednot, ki jih je določil *The Agile Manifesto* [2, 3] in predstavljajo pomemben vidik pri agilnem razvoju programske opreme.

Metodologija ABRD je sestavljena iz aktivnosti, katere združuje v faze in tako omogoča iterativen razvoj. Aktivnosti se v fazi ciklično ponavljajo toliko časa,

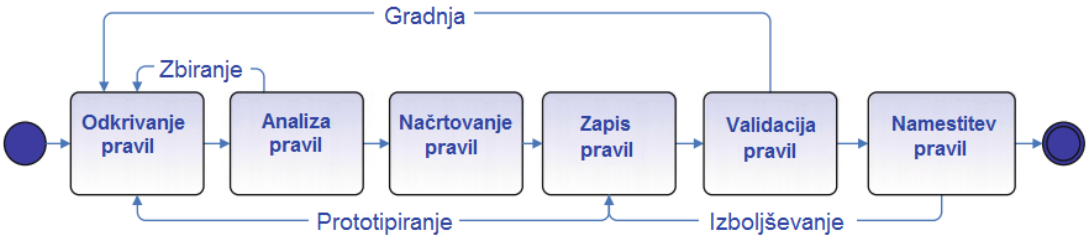
dokler niso izpolnjeni vsi zastavljeni cilji posamezne faze. To pomeni, da se tekom procesa razvoja vsaka aktivnost skoraj zagotovo ponovi večkrat. Aktivnosti metodologije ABRD so:

- odkrivanje pravil (angl. Rule Discovery);
- analiza pravil (angl. Rule Analysis);
- načrtovanje pravil (angl. Rule Design);
- zapis pravil (angl. Rule Authoring);
- validacija pravil (angl. Rule Validation);
- namestitev pravil (angl. Rule Deployment).

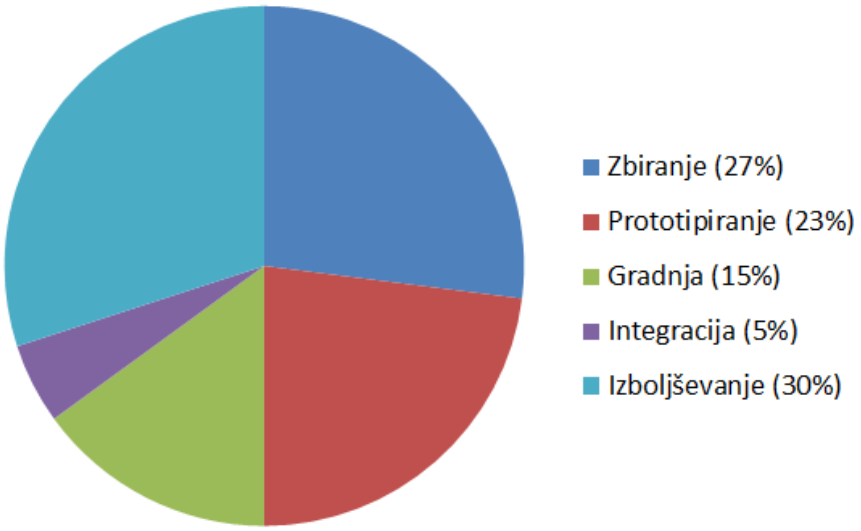
Faze razvoja metodologije ARBD:

- zbiranje (angl. Harvesting);
- prototipiranje (angl. Prototyping);
- gradnja (angl. Building);
- integracija (angl. Integrating);
- izboljševanje (angl. Enhancing).

Slika 2.4 prikazuje razpored aktivnosti v posamezne faze. Krajše faze nam omogočajo, da se poslovna pravila kar najbolje prilegajo poslovnim potrebam. Slika 2.5 prikazuje približen delež časa razvoja, predvidenega za posamezno fazo.



Slika 2.4: Življenjski cikel razvoja poslovnih pravil po metodologiji ABRD [26].



Slika 2.5: Ocena časa, ki naj bi ga porabili v posamezni fazi metodologije ABRD [6].

Faza zbiranja je sestavljena iz odkrivanja in analize poslovnih pravil. V njej zberemo vse dosegljive vire znanja. Viri so lahko opisi poslovnih procesov, priročniki, intervjuji domenskih strokovnjakov itd. Eden glavnih ciljev te faze je poleg zbranih in analiziranih pravil tudi razumeti objektni model domene znotraj obsega razvijajoče aplikacije.

Faza prototipiranja razširja prvo fazo z načrtovanjem in zapisom pravil. V tej fazi se definira struktura projekta, v kateri bodo implementirana poslovna pravila. Definira se predvidene vhodne in izhodne objekte ter zaporedje pravil.

Faza izgradnje pravil zajema implementacijo pravil v končni sintaksi kot tudi testiranje s strani poslovnih uporabnikov. Če je izvedljivo, poslovna pravila namestimo na izvajalno okolje, kjer na podlagi izvajanja izvedemo testiranje. Rezultati izvedenih dinamičnih testov so vredni več kot statičen pregled pravil. Priporočljivo je, da v tej fazi ustvarimo tudi nabor testnih primerov za implementirana poslovna pravila.

V fazi integracije naložimo poslovna pravila na produkcijsko izvajalno okolje ter opravimo test uporabe znotraj predvidenega okolja. Potrebna je uskladitev

oz. transformacija med objektnim modelom poslovnih pravil ter objektnim modelom aplikacij, ki uporabljajo poslovna pravila. Zagotoviti je potrebno pravilno usmeritev zahtevkov ter odgovorov z rezultati izvajanja poslovnih pravil. Nabor testov, ki je bil implementiran v predhodni fazi, vgradimo v primerno programsko infrastrukturo, ter izvedemo ustrezno integracijsko testiranje.

Faza izboljševanja je namenjena dodatnemu testiranju izvajanja pravil ter poganjanju stresnih testov, kar posledično pripelje do izboljšav. Cilj te faze je, da so poslovna pravila implementirana na način, ki bo omogočal njihovo najbolj učinkovito izvajanje. Ob koncu te faze se poslovna pravila premaknejo v dolg cikel upravljanja v produkcijskem okolju.

2.5 Umestitev sistema BRMS v arhitekturo informacijskega sistema

Da lahko v polnosti izkoristimo vse prednosti uporabe sistema BRMS, je potrebna pravilna umestitev slednjega v arhitekturo poslovnega informacijskega sistema. Glede na trenutne potrebe zagotavljanja podpore poslovanja podjetij, najboljše končne produkte omogoča souporaba sistemov BRMS in BPMS ter koncepta SOA (Service Oriented Architecture) [5, 29].

SOA [23, 28] je arhitekturni koncept, ki temelji na uporabi storitev in je neodvisen od specifičnih protokolov ali ponudnikov. Storitev je aplikacijska komponenta, ki implementira neko zaključeno funkcionalnost in jo izpostavlja preko protokolov za izmenjavo podatkov (SOAP, REST ipd.). Tak način izmenjave podatkov pride v poštev pri povezovanju B2B (Business to Business), ker je omogočena komunikacija med neodvisnima storitvama, ki sta implementirani s pomočjo različnih tehnologij (npr. J2EE in .NET). Ena glavnih prednosti tega arhitekturnega koncepta je tudi šibka sklopljenost. Implementacija storitve je neodvisna od programskega vmesnika, ki ga izpostavlja drugim storitvam ali aplikacijam. Razvijalci lahko gradijo aplikacije s kombiniranjem teh storitev, ne da bi natančno poznali njihovo implementacijo. Naslednja pomembna lastnost koncepta SOA je ponovna uporaba, saj lahko isto storitev uporabimo v več drugih aplikacijah. S tem se lahko močno skrajša razvoj nove storitve ali aplikacije in olajša njihovo vzdrževanje. Sistem BRMS lahko v konceptu SOA uporabimo na več načinov. Prvi način je, da

vsaka storitev ločeno dostopa do poslovnih pravil. Drugi (priporočljiv) način pa je, da implementiramo ločeno storitev, preko katere vse ostale storitve dostopajo do poslovnih pravil sistema BRMS. Na ta način dobimo enotno točko dostopa. Ne glede na način dostopa nam sistem BRMS skupaj s konceptom SOA omogoča ponovno uporabo poslovnih pravil ne glede na tip storitve, kar pripelje do lažjega upravljanja in dodatno pospeši razvoj novih aplikacij.

Sistemi BPMS [11] so programski produkti, ki nam omogočajo modeliranje poslovnih procesov, avtomatizacijo človeških opravil (angl. Human Task) in integracijo različnih poslovnih sistemov. V pomoč so nam pri odkrivanju različnih metrik, ki so lahko podlaga za optimizacijo poslovnih procesov. Model poslovnega procesa zajema vse aktivnosti, osebe (vloge) in različne vire (npr. dokumenti), ki jih te osebe uporabljajo v poslovnem procesu. Sistem BRMS služi sistemu BPMS kot vir poslovnih pravil, s katerimi lahko določamo smer toka poslovnega procesa ali le dopolnimo oz. spremenimo določene podatke, ki jih vsebuje poslovni proces. Priporočljivo je, da sistem BPMS uporablja isto implementacijo poslovnih pravil kot ostale aplikacije. S tem se ohranjata konsistentnost pravil ter možnost njihove ponovne uporabe.

Uporaba sistema BPMS ter razvoj programske opreme na podlagi SOA koncepta ne predstavlja pogoja za uspešno integracijo sistema BRMS, v določenih primerih pa lahko pa k njej bistveno pripomoreta.

V naslednjem poglavju predstavimo poslovna pravila. Podamo nekaj definicij, predstavimo dve klasifikaciji ter opišemo najbolj pogoste načine zapisa. Na koncu predstavimo še postopke zajema poslovnih pravil iz zapuščinskih sistemov.

Poglavje 3

Poslovna pravila

Ljudje smo vsakodnevno soočeni z veliko odločitvami. Večinoma se jih niti ne zavedamo in se nanje odzovemo refleksno. Ne glede na težo pa je odločitev rezultat informacij o določeni situaciji in preteklih izkušenj oz. znanju. Podobno velja tudi za poslovno okolje. Večina velikih podjetij se ukvarja z upravljanjem znanja [19] (angl. Knowledge Management). Ta obsega zajemanje, razvijanje, deljenje ter učinkovito uporabo znanja. Dodatno pridobljeno znanje zaposlenim omogoča sprejemanje boljših poslovnih odločitev. Zbrano znanje v podjetju lahko imenujemo tudi poslovna pravila, saj vsebuje direktive in vodila za učinkovito poslovanje podjetja.

Dinamično dogajanje na trgu od podjetij zahteva nenehno prilagajanje svojih poslovnih pravil. Poslovna pravila morajo biti zbrana v centralnem repozitoriju, ki je dostopen vsem zaposlenim. Za vsako poslovno pravilo mora obstajati zgodovina sprememb ter opombe, zakaj je poslovno pravilo nastalo ali bilo spremenjeno. Poslovna pravila morajo biti zapisana tako, da so razumljiva vsem zaposlenim in da jih lahko ti brez večjega napora spreminjajo in dodajajo.

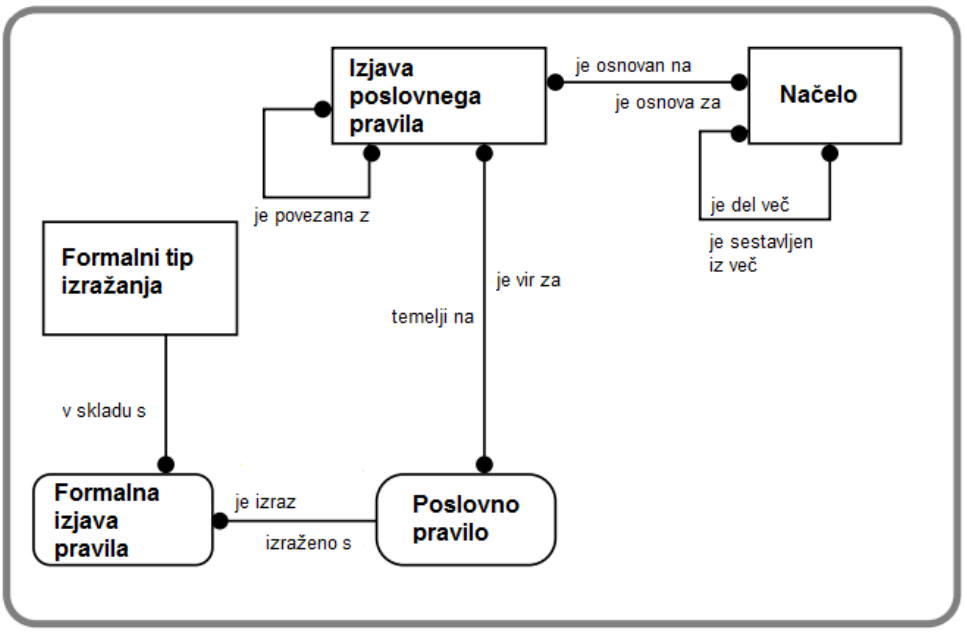
V tem poglavju podamo nekaj definicij poslovnih pravil, predstavimo dve klasifikaciji, opišemo najbolj pogoste načine zapisa in predstavimo postopke zajema poslovnih pravil iz zapuščinskih sistemov.

3.1 Definicija in klasifikacija

Obstaja veliko definicij poslovnih pravil, ki se razlikujejo glede na obdobje nastanka ter uporabniški vidik avtorja. Prve definicije enačijo poslovna pravila z omejitvami v podatkovni zbirki [28]. Daniel S. Appleton [24] jih opredeli kot *eksplicitne izjave, ki opisujejo omejitve znotraj poslovne ontologije.* Ontologija je v informacijski znanosti formalno ime za poimenovanja in definicije tipov, lastnosti in povezovanja entitet, ki so značilne za določeno poslovno domeno. Ronald G. Ross [33] definira poslovno pravilo kot *natančno pravilo ali načelo, ki vodi obnašanje podjetja in ga ločuje od ostalih.* Nekaj let kasneje jih opiše tudi kot *samostojno delujočo prakso ali načelo. Kot poslovno pravilo lahko obravnavamo tudi uporabniške zahteve, ki so izražene v ne proceduralni in netehnični obliki. Poslovno pravilo predstavlja izjavo o poslovnem obnašanju* [34]. Člani skupine BRG [1] poslovna pravila opredelijo kot *izjave, ki definirajo ali omejujejo določene vidike poslovanja. Namenjena so utrjevanju poslovne strukture, upravljanju oz. vplivanju na obnašanje poslovanja. Poslovna pravila morajo biti atomarna. To pomeni, da se ne morejo deliti, ne da bi pri tem izgubili pomembno informacijo.* Tony Morgan [32] o poslovnih pravilih zapiše, da so to *jedrnate izjave o vidiku poslovanja, ki morajo biti izražene s pojmi, ki so neposredno povezani s poslovno domeno, z uporabo jasnega in nedoumnega jezika, ki je dostopen vsem udeležencem.* Ne glede na razlike pa je vsem definicijam skupno to, da govorijo o določilih ali omejitvah poslovanja.

Za predstavitev konceptualnega modela in strukture formalizma poslovnih pravil lahko uporabimo diagram E-R (angl. Entity Relationship) [1]. Predstavljen je na sliki 3.1.

Na desni strani diagrama vidimo *načelo* (angl. Policy), ki predstavlja usmeritev oz. vodilo podjetja. Vsako načelo je lahko sestavljeno iz več načel ali je del več načel. Primer načela je npr.: Podjetje lahko oddaja v najem samo avtomobile, ki so v skladu z zakonom o motornih vozilih. Vsako načelo je lahko osnova za *izjavo poslovnega pravila* (angl. Business Rule Statement) in izjava poslovnega pravila je lahko zasnovana na enem ali več načelih. Izjava poslovnega pravila je deklarativna izjava o določeni strukturi ontologije ali omejitvi, na kateri temelji poslovanje podjetja. Lahko je povezana z eno ali več izjavami. Primer take izjave je: Vsi avtomobili morajo biti pregledani ob koncu najema. Izjava poslovnega pravila je lahko vir za eno ali več *poslovnih pravil* (angl. Business Rule). Podobno kot



Slika 3.1: Izvor poslovnega pravila [1].

izjava poslovnega pravila tudi poslovno pravilo definira ali omejuje določen vidik poslovanja. Razlika pa je v tem, da poslovno pravilo ne more biti razdeljeno na bolj podrobna poslovna pravila, saj bi pri tem izgubili določeno pomembno informacijo. Vsako poslovno pravilo je lahko osnovano na eni ali več izjavah poslovnega pravila. Primer: Avtomobil, ki je od zadnjega servisa prevozil več kot 5000 km, mora biti pred ponovnim najemom oddan na servis, predstavlja poslovno pravilo. Poslovno pravilo je lahko izraženo z eno ali več *formalnimi izjavami pravila* (angl. Formal Rule Statement), toda vsaka formalna izjava poslovnega pravila je lahko izraz le enega poslovnega pravila. Formalna izjava pravila je izraz poslovnega pravila v določeni formalni slovnici in mora biti v skladu z določenim *formalnim tipom izražanja* (angl. Formal Expression Type). Primer formalne izjave pravila v strukturirani angleščini:

```
if (km from last service) > 5000 then
        schedule car for service
end if
```

Podobno kot pri definicijah poslovnih pravil obstaja tudi več vrst njihovih klasifikacij. Glede na strogost upoštevanja [38] jih delimo na:

- določila (angl. Mandates);
- načela (angl. Policies);
- vodila (angl. Guidelines).

Določila so javna pravila, ki jih mora podjetje obvezno upoštevati. V nasprotnem primeru bodo sledile določene posledice. Primer določil je upoštevanje državnih zakonov. Načela so pravila, ki jih je potrebno upoštevati, da se držimo strategij, dogovorjenih pravil in ciljev podjetja. Vodila so poslovna pravila, ki veljajo glede na določene okoliščine. Primer je uporaba metodologije glede na dani problem ali uporaba najboljših praks v dani situaciji.

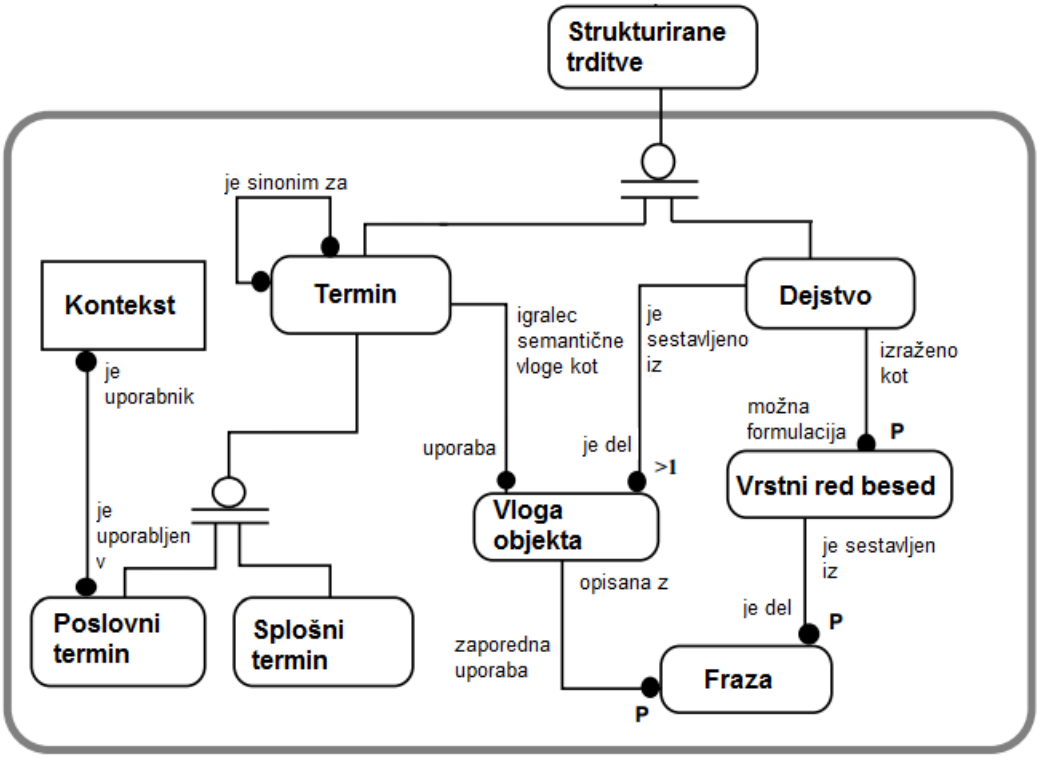
V naslednjih podpoglavjih bomo predstavili drugo delitev, ki poslovna pravila glede na vsebino deli v tri skupine [1].

3.1.1 Strukturirane trditve

Strukturirane trditve (angl. Structural Assertion) so izjave, ki izražajo obstoj določenega koncepta ali pomembnost povezave do nekega drugega koncepta, ki je v interesu podjetja. Opisujejo specifične statične vidike poslovanja in izražajo kako dobro in kateri koncepti se med seboj prilegajo. Strukturirane izjave so velikokrat prikazane z diagramom E-R. Na sliki 3.2 vidimo strukturirane izjave dveh vrst: termini (angl. Terms) in dejstva (angl. Facts).

Termini so besede ali besedne zveze, ki imajo določen pomen za poslovanje. Ločimo *poslovne termine* (angl. Business Terms) in *splošne termine* (angl. Common Terms). Poslovni termin je beseda ali besedna zveza, ki ima določen pomen za poslovanje v sklopu določenega konteksta. Vsak poslovni termin mora biti uporabljen v povezavi z vsaj enim kontekstom. Splošni termini so besede in besedne zveze v vsakdanjem jeziku in imajo splošno sprejet pomen.

Termini, poslovni ali splošni, se delijo na tipe (angl. Type) in dejanske vrednosti (angl. Literal). Delitev prikazuje slika 3.3. Tipi predstavljajo poimenovane abstrakcije skupin primerkov ali vrednosti (npr. stranka, model avtomobila). Posebna vrsta tipa se imenuje senzor (angl. Sensor). Namenjen je opisu zaznave



Slika 3.2: Strukturirane trditve [1].

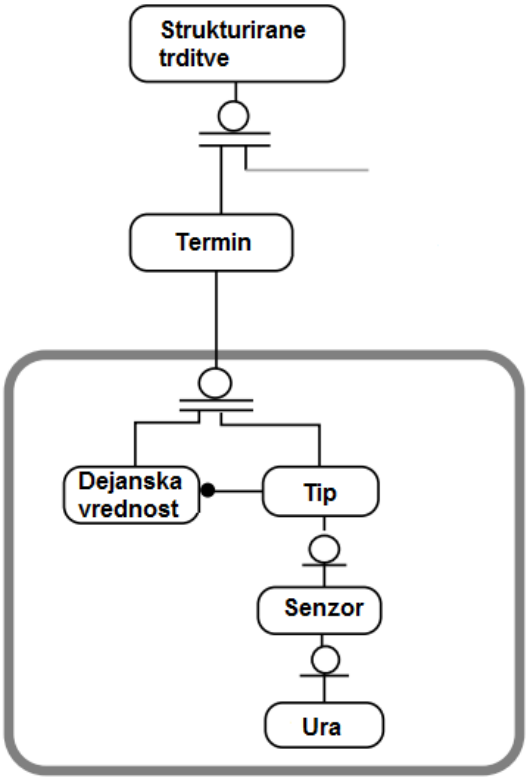
sprememb oz. stanja opazovanega objekta (npr. odčitek termometra). Senzor vsebuje podvrsto ura (angl. Clock), ki poleg lastnosti senzorja vsebuje tudi podatek o času oz. trajanju. Dejanske vrednosti predstavljajo točno določeno vrednost ali instanco tipa.

Dejstva so izjave, ki so sestavljene iz poslovnih in splošnih terminov. Delimo jih na dva načina, ki jih prikazuje slika 3.4.

Glede na način zapisa dejstva delimo na:

- osnovna dejstva (angl. Base Facts);
- izpeljana dejstva (angl. Derived Facts).

Vsako dejstvo je ali osnovno ali izpeljano. Osnovno dejstvo je podano kot konkretna vrednost. Izpeljano dejstvo ima vrednost vedno izračunano ali pa se lahko sklepa glede na neko drugo poslovno pravilo.

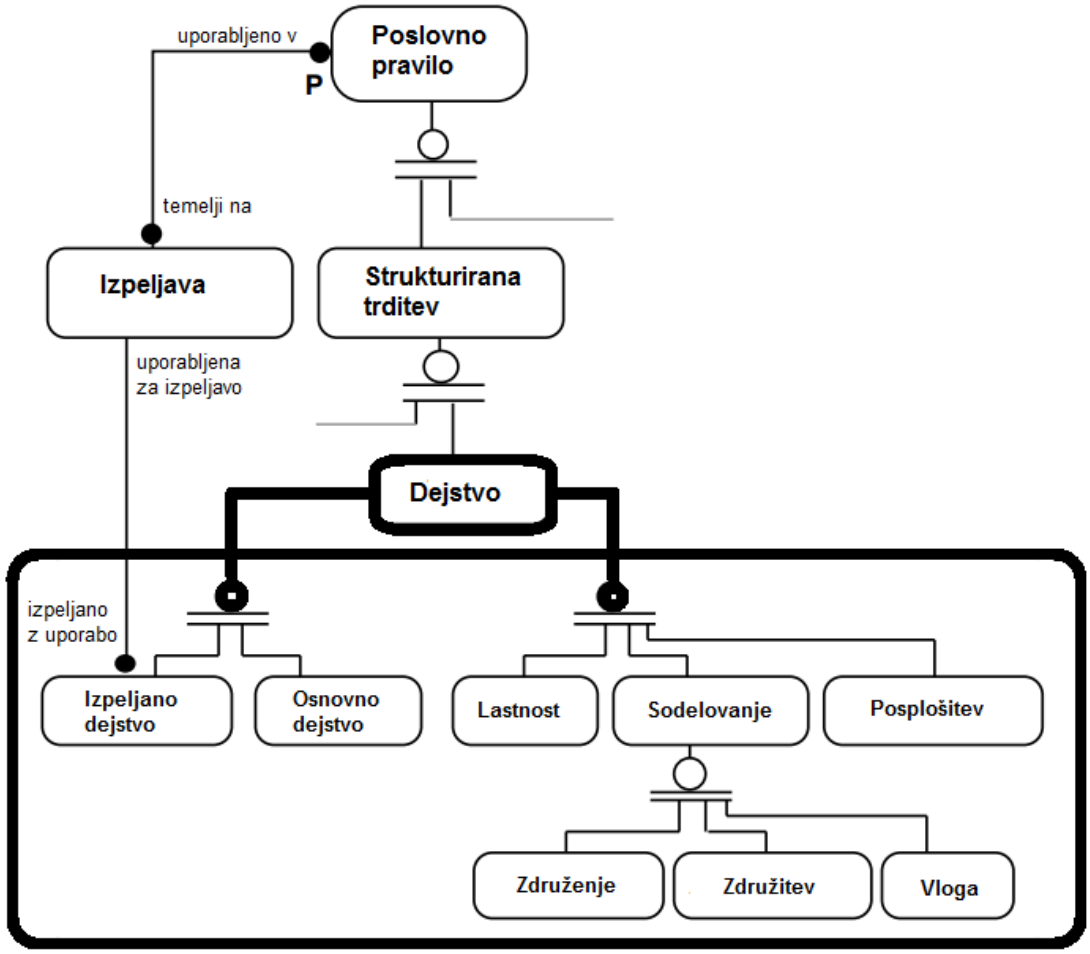


Slika 3.3: Delitev terminov [1].

Klasifikacija dejstev glede na vsebino deli dejstva na:

- lastnosti (angl. Attribute);
- posplošitve (angl. Generalization);
- sodelovanja (angl. Participation), katere naprej delimo na:
  - združenja (angl. Association).
  - združitve (angl. Aggregation);
  - vloge (angl. Role);

Lastnost izraža dejstvo, v katerem termin opisuje lastnost drugega termina (npr. barva je lastnost avtomobila). *Sodelovanje* izraža dejstvo, v katerem je



Slika 3.4: Delitev dejstev [1].

množica terminov med seboj povezana v nekem smislu, ki je pomemben za poslovanje. *Posplošitev* izraža dejstvo, v katerem termin predstavlja nadtip drugega termina (npr. vozilo je nadtip avtomobila). *Združitev* kaže na to, da je neki termin del drugega, oz. da je neki termin sestavljen iz množice drugih. *Vloga* opisuje, kako neki termin služi kot udeleženec pri drugem terminu preko komunikacije s svojim okoljem. Vse ostale posplošitve spadajo v skupino *združenje*.

3.1.2 Akcijske trditve

Akcijske trditve (angl. Action Assertion) so izjave, ki se navezujejo na nek dinamičen vidik poslovanja. Za razliko od strukturiranih izjav, ki opisujejo možnosti, akcijske trditve določajo omejitve, ki predstavljajo rezultat proizvedene akcije. Delimo jih na dva načina. Prvi način razlikuje trditve glede na razred (angl. Class):

- pogoj (angl. Condition);
- integritetna omejitev (angl. Integrity Constraint);
- avtorizacija (angl. Authorization).

Pogoj je trditev, ki v primeru, da je pravilna, lahko v veljavo postavi neko drugo poslovno pravilo. Lahko ga razumemo kot preizkušnjo za druga poslovna pravila, ki bodo zagotovo negativna v primeru, da je to pravilo (pogoj) negativno. Primer pogoja: Ali je avtomobil registriran? *Integritetna omejitev* je trditev, ki mora biti vedno pozitivna. Ima moč takojšnje omejitve vseh akcij, ki bi kršile končno stanje omejitve. Primer: Integritetna omejitev izjavlja, da morajo biti vsi avtomobili registrirani in s tem prepoveduje kreiranje nove instance avtomobila, ki ne bi imel navedene registrske oznake. *Avtorizacija* definira posebne pravice ali privilegije, ki se nanašajo na enega ali več konstruktov (angl. Constructs). Predstavimo jo lahko s predikatom:

(Samo) X lahko dela Y,

pri čemer je X običajno uporabnik, Y pa akcija oz. opravilo, ki je lahko izvršeno. Primer: Oseba z vozniškim izpitom lahko vozi avtomobil.

Drugi način delitve razlikuje akcijske trditve glede na tip:

- aktivator (angl. Enabler);
- časovnik (angl. Timer);
- izvrševalec (angl. Executive).

Aktivator je trditev, ki v primeru pozitivnega stanja dovoljuje oz. omogoči kreiranje objekta, na katerega se izjava navezuje. *Časovnik* je trditev, ki sproži določeno akcijo v primeru presežene določene časovne omejitve. Lahko gre za

časovni interval ali za točno določeno časovno točko. *Izvrševalec* je trditev, ki povzroči ali zahteva izvršitev ene ali več akcij. Primer: Če stranka tri mesece zaostaja s plačilom, se ji zaseže avtomobil. Prvi del (*tri mesece zaostaja*) je časovnik in zahteva nadaljnjo akcijo, drugi del (*zaseže avtomobil*) je tipa izvrševalec.

3.1.3 Izpeljanke

Izpeljanke (angl. Derivation) so poslovna pravila, katerih vrednost se določi iz matematičnega izračuna ali pa se sklepa glede na eno ali več drugih poslovnih pravil. Matematični izračun producira izpeljano dejstvo glede na podani matematični algoritem. Sklep producira izpeljano dejstvo z uporabo logične indukcije in sklepanja. Primer je izračun cene najema vozila. Ta se izračuna glede na izposojeno vozilo, dni najema, vrsto zavarovanja itd.

3.2 Zapis poslovnih pravil

Poslovna pravila morajo biti v sistemu BRMS predstavljena z zapisom, ki omogoča enostavno dodajanje, brisanje ali spreminjanje pravil tehničnim in netehničnim uporabnikom. Zapis mora biti bolj omejujoč kot naravni jezik, brez dvoumnosti in nejasnosti. Priporočljiva je uporaba posebnega jezika za zapis poslovnih pravil (angl. Business Rule Language). Zapisi se v splošnem delijo na tekstovna in vizualna.

Tekstovni zapisi so običajno zelo podobni naravnemu jeziku. Od naravnega se razlikujejo v tem, da tekstovni zapis omejuje naravni jezik tako, da lahko tvorimo le trditve določene oblike, v katerih so uporabljene posebne rezervirane besede. Definirana oblika trditve se imenuje vzorec (angl. Pattern). Rezervirane besede pa so npr. če, potem, mora ipd. Primer: Status kupca se mora nastaviti na *gold*, če je v zadnjem letu opravil več kot 10 nakupov.

Vizualni jeziki prikazujejo poslovna pravila z različnimi vizualnimi gradniki, ki so med seboj povezani. Predstavljajo lahko dejstvo, termin, omejitev, dejanje itd. Pri tem je pomembno predvsem to, da je pomen vsakega gradnika in povezave natančno definiran.

V nadaljevanju so predstavljeni zapisi poslovnih pravil, ki so zaradi načina zapisa največkrat uporabljeni v sistemih BRMS:

- **Produkcijska pravila** (ang. production rules). Velikokrat se uporablja tudi ime *pogojno akcijska pravila* (angl. Condition-Action Rules). Splošen zapis ima obliko:

  ```
  if A then X,
  ```

  kjer A predstavlja pogoj, X pa veljaven sklep ali posledico. Zapis lahko tolmačimo na način: *Če smo zadostili določenim pogojem, naj se zgodi določena akcija* ali *Če je pogoj A resničen, potem velja X.*

  Pogoji so lahko enostavni ali sestavljeni iz več preprostih pogojev in povezani z operatorji in (angl. And), ali (angl. Or) in ne (angl. Not).

  Produkcijska pravila so enostavna za razumevanje, saj vsako vsebuje le majhen in neodvisen del zbranega znanja. Ker so pravila med seboj neodvisna, omogočajo deklarativen zapis, obenem pa so tudi preprosta za urejanje, dodajanje in brisanje. Zaradi teh lastnosti so bila že v preteklosti jedro aplikacij za podporo odločitvam (DENDRAL [31], MYCIN [27, 20], PROSPECTOR [30], itd.) in predstavljajo osnovo za skoraj vse moderne sisteme BRMS.

- **Odločitvene tabele** (angl. Decision Table) uporabljamo za zapis poslovnih pravil, ki imajo večino pogojev definiranih nad istimi tipi ali atributi. Predstavljajo pregleden in jedrnat zapis poslovnih pravil. Glede na način vnosa pogojev, akcij in podatkov ločimo odločitvene tabele z razširjenim (angl. Extended Entry Decision Table) in omejenim vnosom (angl. Limited Entry Decision Table).

  Pri odločitvenih tabelah z razširjenim vnosom v stolpcih definiramo omejitve na objektu ali njegovem atributu ter akcijo ali sklep, ne pa tudi vrednosti. Te so podane v vrsticah odločitvene tabele. Primer predstavlja slika 3.5. Za razliko od tabel z razširjenim vnosom pa definicija stolpcev odločitvene tabele z omejenim vnosom vsebuje tudi vrednost. V vsaki vrstici označimo, kateri pogoji in akcije veljajo za posamezno poslovno pravilo. Primer predstavlja slika 3.6. Ne glede na vrsto odločitvene tabele vsaka vrstica predstavlja eno poslovno pravilo.

status uporabnika	znesek nakupa [€]	popust pri nakupu [%]
navaden	< 100	0
navaden	100-300	1
navaden	> 300	3
silver	< 100	0
silver	100-300	2
silver	> 300	4
gold	< 100	0
gold	100-300	3
gold	> 300	5

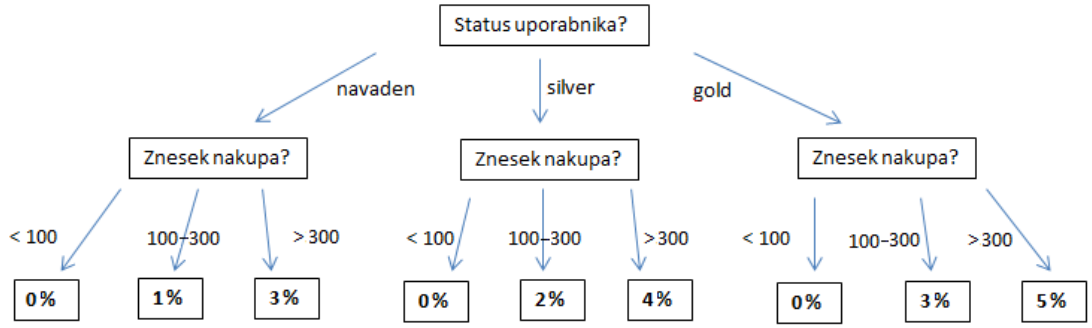
Slika 3.5: Odločitvena tabela z razširjenim vnosom.

status == navaden	status == silver	znesek < 100€	100€ < znesek < 300€	popust 0%	popust 1%	popust 2%
☑	☐	☑	☐	☑	☐	☐
☑	☐	☐	☑	☐	☑	☐
☐	☑	☑	☐	☑	☐	☐
☐	☑	☐	☑	☐	☐	☑

Slika 3.6: Odločitvena tabela z omejenim vnosom.

- **Odločitvena drevesa** (angl. Decision Tree) so vizualna predstavitev poslovnih pravil. Primer predstavlja slika 3.7. Odločitveno drevo je sestavljeno iz korenskega vozlišča, podvozlišč in medsebojnih povezav. Iz vsakega vozlišča izhaja ena ali več povezav do podvozlišč, razen v primeru končnega vozlišča. Vozlišče predstavlja pogoj, povezava pa vrednost (ali zalogo vrednosti), ki jo lahko atribut zavzema. Če je pogoj v povezavi izpolnjen, se po njej premaknemo v naslednje vozlišče. Končno vozlišče predstavlja akcijo oz. posledico, ki sledi glede na izpolnjene pogoje. Vsaka pot od korenskega do končnega vozlišča predstavlja eno poslovno pravilo.

  Odločitveno drevo predstavlja zelo pregleden in izčrpen prikaz poslovnih pravil, ki so med seboj povezana oz. imajo veliko skupnih pogojev. Če poslovna pravila nimajo veliko skupnih pogojev, drevo postane zelo razvejano in posledično nepregledno.



Slika 3.7: Odločitveno drevo s pravili za določanje popusta glede na status uporabnika in znesek nakupa.

- **Naravni jezik domene** oz. jezik DSL (angl. Domain-Specific Language) je jezik, ki omogoča pisanje programskih stavkov v naravnem jeziku določene problemske domene [35]. Ti programski stavki se glede na definicijo jezika DSL transformirajo v obliko, ki je primerna za prevajanje ali neposredno za izvajanje.

  Jeziki DSL so običajno deklarativni ter imajo majhen nabor ukazov ali pa širijo neki splošno namenjen jezik (angl. General-Purpose) s funkcionalnostmi namenjenimi točno določeni problemski domeni.

  Danes obstaja veliko splošno uporabljenih jezikov DSL [12]. Orodja Matlab ter GNU Octave sta jezika DSL namenjena računanju z matrikami (angl. Matrix Programming). Jezika Verilog in VHDL sta namenjena le načrtovanju oz. opisovanju delovanja strojne opreme (angl. Hardware Description Language).

  V okviru poslovnih pravil nam definicija jezika DSL omogoča, da so poslovna pravila napisana v naravnem jeziku poslovne domene oz. v žargonu domene, ter so tako še bolj razumljiva kot produkcijska pravila.

  Uvedba jezika DSL prinaša tudi določene slabosti ter stroške. Potrebno ga je načrtovati ter implementirati. S slabo definicijo jezika DSL ne moremo implementirati preglednih ter nedvoumnih poslovnih pravil. Poleg upravljanja s poslovnimi pravili se je potrebno ukvarjati tudi z upravljanjem samega

jezika DSL. Glede na potrebe poslovne domene ga je potrebno popravljati ter dopolnjevati. Kot vsak drug ima tudi jezik DSL določeno sintakso, ki se jo morajo uporabniki naučiti, če ga želijo pravilno uporabljati.

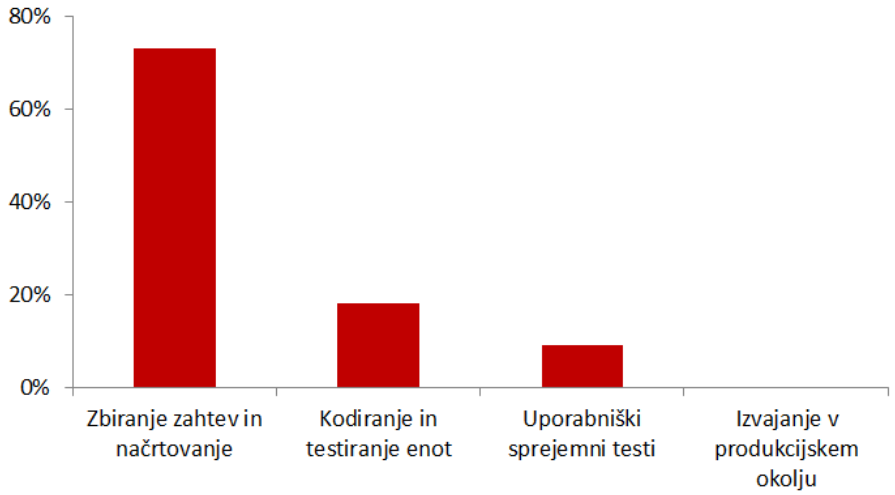
Večina implementacij sistemov BRMS ponuja uporabniku prijazen uporabniški vmesnik za kreiranje in urejanje vsake od zgoraj naštetih oblik za potrebe zapisa poslovnih pravil. Ne glede na izbrano obliko zapisa poslovnih pravil se ta pred namestitvijo na izvajalno okolje transformira v enotno obliko oz. sintakso (pogosto kar v produkcijska pravila). Tako omogočimo izvajalnemu okolju, da lahko razume le eno sintakso zapisa poslovnih pravil.

3.3 Pridobivanje poslovnih pravil iz zapuščinskih sistemov

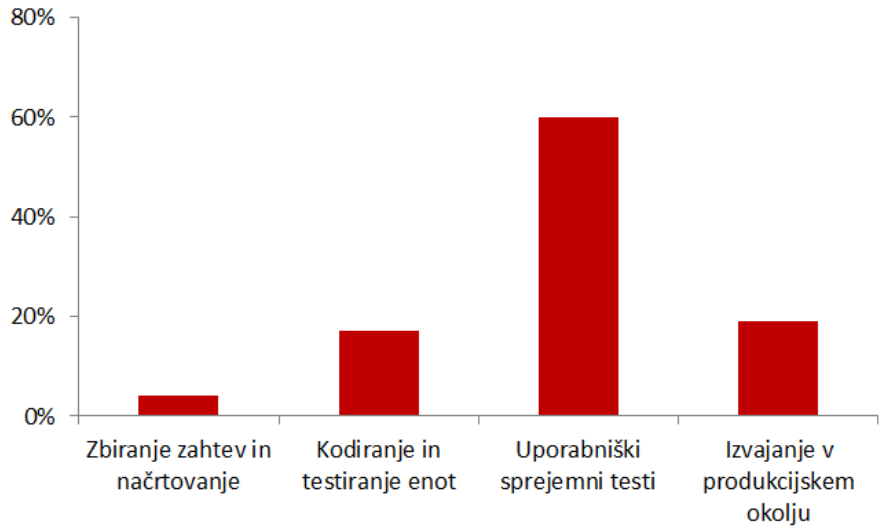
Tako kot večina programskih produktov so tudi poslovni informacijski sistemi v svojem življenjskem ciklu deležni nadgradenj in posodobitev. Velikokrat se zgodi, da obstoječega informacijskega sistema zaradi arhitekture ali tehnologije ni več mogoče nadgraditi ali pa se dolgoročno ne izplača in ga je zato potrebno zamenjati. Ne glede na obseg modernizacije sistema pa se je potrebno držati načela, da se poslovna pravila ne smejo spremeniti. Ta se spremenijo le, ko se spremenijo pravila poslovanja podjetja.

Preden se lotimo ekstrakcije pravil iz sistema, ki ga bomo zamenjali, je potrebno oceniti tveganje modernizacije [7] ter oceniti posledice, ki jih lahko prinese napačno odkrito ali ne odkrito poslovno pravilo. Ocena naj bo vodilo pri vzpostavitvi minimalne natančnosti ekstrakcije poslovnih pravil.

Podobno kot z ostalo programsko kodo je tudi pri poslovnih pravilih največja možnost vnosa napak v fazi zbiranja zahtev ter načrtovanja. Približne deleže vnosov napak v posameznih fazah razvoja poslovnih pravil nam prikazuje slika 3.8. Največ napak se odkrije v fazi uporabniškega testiranja. Odkritje oz. odpravljanje napake v kasnejših fazah je lahko za podjetje 100-krat dražje [4] z vidika poslovnih virov kot v fazi načrtovanja. Približne deleže odkritja napak v posameznih fazah razvoja poslovnih pravil nam prikazuje slika 3.9.



Slika 3.8: Deleži vnosov napak v posamezni fazi razvoja poslovnih pravil [7].



Slika 3.9: Deleži odkritja napak v posamezni fazi razvoja poslovnih pravil [7].

3.3.1 Metode ekstrakcije poslovnih pravil

Tipična predpostavka pri načrtovanju poslovnih sistemov je, da so poslovni analitiki sposobni izdelati popoln nabor poslovnih pravil samo iz lastnega znanja in zabeleženih specifikacij. Pri velikih sistemih je to nemogoče pričakovati. Zato je po [7] predlagana tristopenjska metoda pridobivanja pravil, sestavljena iz naslednjih korakov:

- **Klasična poslovna analiza** zajema pogovore s poznavalci stroke ter pregled dokumentov, ki vsebujejo podatke o načinu poslovanja podjetja. Na ta način lahko pričakujemo od 50 do 70 odstotkov uspešno pridobljenih poslovnih pravil. Pri analizi pridobljenih pravil v tej fazi se je potrebno zavedati, da bo vsaj nekaj odstotkov pravil napačnih ali zelo dvoumnih. Te je potrebno s pomočjo primerjave rezultatov naslednje analize ter domenskih strokovnjakov v nadaljevanju postopka odpraviti.

- **Statična ekstrakcija pravil** predstavlja uporabo programskih orodij, ki analizirajo programsko kodo. Njihovo delovanje je podobno prevajalniku oz. interpreterju, le da rezultat ni izvedljiva koda, temveč repozitorij, ki poleg drugih informacij vsebuje tudi tehnično zapisana poslovna pravila. Ta tehnični zapis je bolj podoben zakodiranim poslovnim pravilom v programski kodi kot zapisu, ki bi bil primeren za poslovnega analitika. Vsa ta tehnično zapisana pravila je potrebno pregledati, kar je vseeno lažje kot neposredni pregled programske kode. Problem uporabe te metode je, da zbirka poslovnih pravil, pridobljena kot rezultat, vsebuje tudi pravila, ki niso več v uporabi. Ob koncu te faze lahko pričakujemo, da se nivo najdenih poslovnih pravil dvigne na 80 do 90 odstotkov.

- **Dinamična ekstrakcija pravil** predstavlja uporabo programskih orodij, ki analizirajo način izvajanja programa ter orodja za odkrivanje deleža izvedene kode (angl. Code Coverage Analysis Tools). Ta orodja prikažejo, kateri programski ukazi so se dejansko izvedli in kateri ne (t.i. mrtva koda) glede na nabor vhodnih podatkov. Del dinamične ekstrakcije pravil je tudi izgradnja testnih primerov iz že pridobljenih poslovnih pravil. Te teste skupaj s starim sistemom namestimo v kontrolirano izvajalno okolje. Analiza

rezultatov testiranja nam pokaže, ali so do sedaj pridobljena poslovna pravila pravilna ali ne. Po končani dinamični ekstrakcij pravil ter njihovi analizi lahko pričakujemo, da smo uspešno odkrili vsa poslovna pravila.

Meja med fazami ni strogo določena. Prehod iz trenutne v naslednjo fazo izvedemo, ko hitrost pridobivanja novih pravil v trenutni fazi drastično pade ali ko smo prepričani, da bo izvajanje naslednje faze prineslo hitrejši napredek pri odkrivanju pravil. Rezultati posamezne faze se med seboj dopolnjujejo ter preverjajo. Najdene razlike ali dvoumnosti je potrebno analizirati ter odpraviti.

Testne primere, ki smo jih zgradili v zadnji fazi, je priporočljivo izvesti tudi na moderniziranem oz. novem sistemu. Če se semantično ne ujemajo vsi rezultati testov pri starem in novem sistemu, je to pokazatelj napake pri implementaciji. Tudi v primeru, da se vsi rezultati semantično ujemajo, to še ni zagotovilo, da je novi sistem brez napak, saj obstaja verjetnost, da testi ne pokrivajo vseh možnih scenarijev delovanja oz. uporabe sistema. Za zagotavljanje popolne ustreznosti sistema je priporočljiva tudi uporaba orodja, ki ugotavlja delež izvedene programske kode. V primeru, da se ne izvedejo vse vrstice kode novega sistema pri polnem naboru vhodnih podatkov, je potrebo pregledati nenamerno vneseno funkcionalnost ter jo glede na rezultate analize odpraviti.

Ne glede na kakovost specifikacij in celovitost poslovnih pravil, ki smo jih pridobili, je edino merilo ustreznosti sistema njegovo izvajanje.

Z zavedanjem pomena sistemov BRMS se je razširila ponudba produktov na trgu. V naslednjem poglavju predstavimo dva komercialna ter dva odprtokodna produkta.

Poglavje 4

Produkti BRMS

Z zavedanjem podjetij o prednostih uporabe sistemov BRMS se je pospešil njihov razvoj ter razširila ponudba. Veliko število programskih platform in ekosistemov podjetij je omogočilo razvoj velikega števila različnih produktov. Najpomembnejši dejavnik pri izbiri sistema BRMS je možnost njegove integracije z informacijskim sistemom podjetja ter celotna cena njegove uporabe (angl. Total Cost Of Ownership). Vedno več podjetij se odloča za odprtokodne sisteme BRMS, saj so že dovolj razviti za uporabo tudi v velikih poslovnih sistemih ter ne zahtevajo velike začetne investicije. Za večino od njih obstaja tudi komercialna podpora. V nadaljevanju poglavja sta predstavljena dva komercialna sistema BRMS ter dva odprtokodna sistema BRMS. Poleg naštetih obstajajo tudi SAP NetWeaver BRM, Oracle Business Rules, Pega Business Rules Platform, Fico Blaze Advisor, FlexRule in drugi.

4.1 IBM ODM

IBM ODM (Operational Decision Management) [10] je platforma za zajem, avtomatizacijo in upravljanje poslovnih odločitev. Leta 2009 je IBM prevzel podjetje ILOG, ki je v svoji zbirki produktov ponujalo tudi sistem BRMS JRules. Po prevzemu se je produkt preimenoval v IBM Websphere ILOG JRules, leta 2015 pa v IBM ODM. Zadnja verzija platforme je 8.5.1. Sestavljena iz dveh večjih modulov [17]:

- **IBM Decision Center** je centralni repozitorij za shranjevanje ter urejanje

poslovnih pravil. Za upravljanje pravil ponuja dva vmesnika oz. konzoli (angl. Console). *Business* konzola je namenjena poslovnim uporabnikom, saj ponuja prilagojen uporabniški vmesnik za upravljanje s poslovnimi pravili. *Enterprise* konzola je namenjena zahtevnejšim uporabnikom, saj poleg urejanja poslovnih pravil, ponuja tudi administrativne funkcionalnosti. Ponuja tudi vtičnike, ki omogočajo urejanje poslovnih pravil iz različnih urejevalnikov besedil, kot sta MS Word in MS Excel.

- **IBM Decision Server** je izvajalno okolje za poslovna pravila in poslovne dogodke. Vsebuje tudi orodja za razvoj in implementacijo poslovnih pravil ter modeliranje poslovnih procesov. V večjih poslovnih okoljih je za razvoj in implementacijo priporočljiva uporaba orodja IBM Decision Center, saj ponuja bolj primeren uporabniški vmesnik za poslovne uporabnike.

Platforma vsebuje tudi manjšo komponento IBM Decision Server Rules Edition for Integration Bus za lažjo integracijo platforme IBM ODM z integracijskim vodilom IBM Integration BUS. Podjetje IBM ponuja tudi platformo IBM ODM Express, ki vsebuje manj funkcionalnosti kot polna platforma IBM ODM. Primerna je za manjša podjetja, ki šele uvajajo uporabo odločitvenih sistemov, saj ni tako obsežna in zahteva manjšo začetno investicijo.

Platforma IBM ODM je le ena izmed komponent velikega ekosistema IBM. Velikokrat se uporablja skupaj z orodjem IBM BPM, ki je ena vodilnih rešitev za upravljanje poslovnih procesov.

4.2 InRule

Platforma InRule [15] je družina produktov za upravljanje in izvajanje poslovnih pravil podjetja InRule Technology [16]. Prva različica je izšla leta 2002, ko je bilo podjetje ustanovljeno. Trenutno aktualna različica InRule 5 je izšla leta 2016 in zajema komponente:

- **irAuthor** je okolje za upravljanje s pravili. Ponuja preprost in intuitiven uporabniški vmesnik, ter tako omogoča tehničnim in netehničnim uporabnikom hitro dodajanje in urejanje pravil.

- **irWord** je vtičnik za urejevalnik besedila MS Word, ki omogoča pisanje poslovnih pravil. Ponuja vse funkcionalnosti komponente irAuthor.
- **irStudio** je vtičnik za razvojno orodje MS Visual studio, ki omogoča pisanje poslovnih pravil. Ponuja vse funkcionalnosti komponente irAuthor.
- **irVerify** je komponenta za preverjanje pravilnosti poslovnih pravil. Omogoča kreiranje testnih scenarijev za posamezna pravila ali skupino pravil ter generiranja poročila izvajanja, v katerem so navedene podrobnosti ovrednotenja, razvrščanja ter izvajanja poslovnih pravil.
- **irCatalog** omogoča centralizirano shranjevanje ter upravljanje poslovnih pravil. Omogoča tudi upravljanje s pravicami, s katerimi določimo omejitev dostopa le avtoriziranim uporabnikom. Zagotavlja beleženje vseh sprememb pravil ter omogoča restavriranje pravil na starejšo različico. Za shranjevanje pravil uporablja podatkovno zbirko MS SQL Server oz. Oracle.
- **irServer** je izvajalno okolje poslovnih pravil. Okolje omogoča integracijo z aplikacijami, ki temeljijo na platformi .NET, ali je nameščeno kot ločena storitev, ki izpostavlja svoje funkcionalnosti preko vmesnika SOAP oz. REST.
- **irSDK** je zbirka aplikacijskih vmesnikov, ki omogočajo InRule integracijo z aplikacijami, ki temeljijo na platformi .NET.

InRule ponuja celostno paleto orodij za upravljanje poslovnih pravil kot tudi za njihovo učinkovito izvajanje. Najlažje se integrira s tehnologijami podjetja Microsoft vključno z oblačno storitvijo Microsoft Azure. Zaradi možnosti izpostavitve izvajalnega okolja v obliki spletnih storitev je primeren tudi za integracijo z ostalimi tehnologijami (Java, PHP, JavaScript itd.).

4.3 OpenRules

Orodje OpenRules [21] je splošno namembni sistem za upravljanje s poslovnimi pravili ter odločitvami (angl. Business Rules And Decision Management System). Produkt je razvilo podjetje *Intelligent ChoicePoint, Inc.* v začetku leta 2003. Konec istega leta se je podjetje preimenovalo v *OpenRules, Inc.* ter ponudilo omenjeni

produkt na trg. Orodje OpenRules je na voljo pod dvema različnima licencama: GPL licenciranje (angl. General Public License) za odprtokodne projekte ter ne-GPL licenciranje za komercialne projekte. Aktualna različica 6.4.0 je izšla julija 2016 in združuje naslednje komponente:

- **Rules Repository** predstavlja centralni repozitorij pravil, ki je namenjen organizaciji ter povezovanju datotek, v katerih so zapisana poslovna pravila. Podprta sta zapisa pravil v datotekah *xls* in notaciji *xml*. Te datoteke se lahko fizično nahajajo v direktorijski strukturi repozotorija ali pa repozitorij vsebuje le povezavo do datoteke, ki se nahaja v podatkovni zbirki, oddaljenem aplikacijskem strežniku ali sistemu za verzioniranje datotek. Za kreiranje in vzdrževanje pravil lahko uporabimo splošno uporabljena orodja za delo s preglednicami oz. različnimi urejevalniki besedila (MS Excel, OpenOffice Calc, Google Sheets, Notepad itd.).

- **Rule Learner** je komponenta, ki s pomočjo algoritmov za strojno učenje obdela pretekle poslovne podatke in rezultate preteklih ovrednotenj poslovnih pravil. Na podlagi pridobljenih rezultatov generira nova poslovna pravila. Generirana poslovna pravila se lahko samodejno shranijo v repozitorij pravil in so takoj dostopna za uporabo pri naslednjem ovrednotenju poslovnih pravil.

- **Rule engine** oz. *OpenRulesEngine* je okolje za izvajanje poslovnih pravil.

- **Rule solver** je komponenta, ki uporablja principe omejitvenega programiranja (angl. Constraint Programming) za modeliranje ter reševanje razporejanja, načrtovanja, konfiguriranja in optimizacije poslovnih virov.

- **Rule Dialog** je komponenta, ki omogoča netehničnim uporabnikom izdelavo spletnih vprašalnikov le s pomočjo tabel urejevalnika Microsoft Excel.

Projekte OpenRules lahko integriramo v aplikacijo programskega jezika Java ali namestimo kot ločeno spletno storitev. Na ta način uporaba poslovnih pravil ni omejena le na aplikacije programskega jezika Java, temveč jih lahko uporabimo v vseh programskih okoljih, ki omogočajo izvedbo klica spletne storitve.

4.4 Drools

Platforma Drools omogoča integracijo poslovne logike (angl. Business Logic Integration Platform oz. BLIP). Gre za odprtokodni projekt, napisan v programskem jeziku Java, ki ga je razvilo in ga še vedno podpira ter nadgrajuje podjetje JBoss (od leta 2006 divizija podjetja Red Hat). Prva različica je izšla leta 2001, od takrat pa je produkt doživel veliko nadgradenj. Nekatere dodane funkcionalnosti so se sčasoma tako razvile, da so postale samostojni produkti. Zadnja večja različica 6.0.0 je izšla novembra 2013. Produkti iz iste družine so bili združeni pod skupno ime KIE (Knowledge Is Everything).

Produkti, ki so zajeti v družino KIE:

- **OptaPlanner** [22] je sistem za optimizacijo načrtovanja poslovnih virov (angl. Business Resource Planning). Uporabljamo ga lahko za optimizacijo porazdelitve ali izkoriščanja poslovnih virov. Sistem pregleda vse možne rešitve problema in izbere najbolj primerno. Uporaba naprednih hevrističnih in metahevrističnih algoritmov (iskanje z metodo Tabu [angl. Tabu Search], simulirana ojačitev [angl. Simulated Annealing] in pozen sprejem [angl. Late Acceptance]) mu omogoča hiter in učinkovit izračun rezultatov. Orodje OptaPlanner je enostavno načrtovalsko izvajalno orodje (angl. Planning Engine), ki ga vgradimo v svojo aplikacijo.

- **jBPM** [18] je zbirka orodij za upravljanje s poslovnimi procesi (angl. Business Proces Management Suite). Omogoča modeliranje poslovnih ciljev in korakov za njihovo izpolnitev. Celoten postopek je modeliran z diagramom poteka (angl. Flow Chart), kar pripomore k boljši preglednosti in omogoča lažje upravljanje. Zbirka jBPM vsebuje orodja, ki so primerna tako za razvijalce kot tudi za poslovne uporabnike, ter tako zmanjšuje semantične razlike med njimi. Jedro predstavlja stroj za izvajanje procesov (angl. Workflow Engine), ki izvaja poslovne procese po specifikaciji BPMN 2.0. Lahko ga vgradimo v našo aplikacijo ali postavimo samostojno kot ločeno storitev.

- **Drools**

  - **Drools Expert** je okolje za izvajanje poslovnih pravil (angl. Rule Engine). Predstavlja osrednji modul, od katerega so vsi ostali odvi-

sni oz. ga razširjajo. Izvajalno okolje opravi sklepanje (angl. Reason Over) na podlagi predefiniranih poslovnih pravil in vstavljenih dejstev oz. podatkov ter nam na podlagi teh podatkov posreduje rezultat. Za sklepanje uporablja algoritem PHREAK, ki se je razvil iz algoritma RETE. Bistvena prednost PHREAK pred RETE je leno obnašanje (angl. Lazy Behaviour). Ta lastnost zmanjša porabo delovnega pomnilnika in omogoča izvajalnemu okolju obravnavo večjega števila pravil in vstavljenih podatkov. Izvajalno okolje lahko vgradimo v aplikacijo ali ga namestimo kot ločeno storitev.

- **Drools Fusion** je razširitev Drools Expert, ki dodaja funkcionalnost obravnave kompleksnih dogodkov (angl. Complex Event Processing). Dogodek predstavlja vsako spremembo stanja sistema z določenim časovnim zaznamkom - ima trajanje ali zavzema le časovno točko. To poleg kvalitativne omogoča tudi časovno obravnavo dogodkov. Glavna lastnost procesiranja dogodkov je ugotavljanje povezav in vzorcev med njimi ter izbor pomembnejših dogodkov. S tem se avtomatizira odziv na pomembne poslovne dogodke in možnost hitrejše (lahko tudi avtomatizirane) obravnave.
- **Drools Workbench** je spletna aplikacija, ki omogoča upravljanje s poslovnimi pravili. Poslovna pravila lahko zapišemo kot produkcijska pravila, odločitvene tabele, odločitvena drevesa, omogoča pa tudi definicijo jezika DSL in zapis poslovnih pravil v naravnem jeziku domene. Ponuja nam možnost vnosa pravil preko uporabniškega vmesnika, ki nas vodi skozi postopek kreiranja pravil. To precej poenostavi delo netehničnim uporabnikom, saj jim ni potrebno skrbeti za sintakso, ker je ta generirana avtomatsko.

  Orodje Drools Workbench temelji na dveh zelo razširjenih orodjih: Apache Maven [9] in Git [14]. Apache Maven se uporablja za kreiranje, upravljanje in arhiviranje (angl. Build) projekta, v katerem implementiramo poslovna pravila. Arhivirani projekti oz. artefakti se naložijo v lokalni repozitorij artefaktov Maven, kjer se hranijo vse različice artefaktov poslovnih pravil. To nam omogoča, da se v primeru napak na novi različici lahko hitro vrnemo na zadnjo delujočo različico.

Orodje Git se uporablja za nadzor različic vseh datotek v projektu.

- **Strežnik Kie** (angl. Kie Server) [13] je samostojna spletna aplikacija, ki omogoča instanciranje poslovnih pravil in procesov. Svoje funkcionalnosti izpostavlja preko vmesnikov REST in JMS (angl. Java Message Service). Omogoča popolno integracijo z orodjem Drools Workbench. Strežnik Kie temelji na principu razširitev (angl. Extensions). Vsaka razširitev je upravljana neodvisno od drugih. Privzeto sta nameščeni dve:
  - BRM, ki zagotavlja podporo za izvajanje poslovnih pravil z uporabo izvajalnega okolja Drools Expert;
  - BPM, ki zagotavlja podporo za izvajanje poslovnih procesov z uporabo izvajalnega okolja jBPM (jBPM Workflow Engine). Ta podpira izvajanje procesov (angl. Process Execution), izvajanje nalog (angl. Task Execution) in asinhrono izvajanje opravil (angl. Assynchronous Job Execution).

  Privzeto nameščeni razširitvi lahko onemogočimo s pomočjo zagonskih nastavitev. Strežnik Kie je lahko zagnan v dveh različnih načinih:
  - nadzorovan (angl. Managed): v tem načinu strežnik Kie potrebuje nadzornika oz. upravljavca. Upravljavec je odgovoren za pravilen zagon ter hrambo in vzdrževanje nastavitev strežnika Kie. Omogočati mora kreiranje, zaganjanje, ustavljanje in brisanje vsebnikov Kie (angl. Kie Container). Ob zagonu se strežnik Kie poizkuša povezati na definiranega upravljavca. V primeru, da povezovanje ni uspešno se strežnik Kie ne zažene.
  - nenadzorovan (angl. Unmanaged): v tem načinu je strežnik Kie postavljen kot samostojna instanca in ni povezan z nobenim upravljavcem. Sam mora skrbeti za vzdrževanje svojih nastavitev. Izpostavljena ima vmesnika REST in JMS, preko katerih omogoča upravljanje z vsebniki Kie.

Vsebnik Kie instancira poslovna pravila implementirana v projektu Drools. Konfiguracija za instanciranje poslovnih pravil se nahaja v datoteki *kmodule.xml*. V njej na deklarativen način definiramo zbirke Kie (angl. Kie Base) ter znotraj njih seje Kie (angl. Kie Session).

Zbirka Kie predstavlja sklop znanja, definiranega v poslovnih pravilih, funkcijah, podatkovnih tipih itd. Sklop znanja definiramo tako, da določimo, na katere pakete oz. direktorije projekta je vezana določena zbirka Kie. Več zbirk Kie je lahko vezanih na isti paket.

Seja Kie se instancira na podlagi ene ali več zbirk Kie. V njej poteka ovrednotenje poslovnih pravil ter izvajanje poslovnih procesov glede na definirano znanje zbirke Kie, vstavljenih podatkov ter nastavitev seje Kie. Delijo se na seje, ki pomnijo stanje med ovrednotenjem pravil (angl. Stateful Session) in seje, ki ne pomnijo stanja (angl. Stateless Session). Pomnjenje stanja vključuje tudi pomnjenje vstavljenih podatkov, tako da naslednje ovrednotenje poteka na starih in na novo vstavljenih podatkih. V sejah brez pomnjenja stanja se po vsakem ovrednotenju pobriše delovni pomnilnik vstavljenih podatkov. Pred vsakim ovrednotenjem jih je tako potrebno ponovno vzpostaviti. Za pomnjenje in brisanje podatkov sej je odgovoren vsebnik Kie.

Vsebnik Kie je torej odgovoren za ustvarjanje in upravljanje zbirk Kie in sej Kie. V primeru, da eksplicitno ne definiramo nobene zbirke Kie in seje Kie, bo vsebnik Kie ustvaril zbirko Kie in iz nje sejo Kie s privzetimi nastavitvami.

V naslednjem poglavju predstavimo uporabo Drools na praktičnem primeru.

Poglavje 5

Implementacija poslovnih pravil na primeru spletne trgovine

V tem poglavju bomo opisali implementacijo poslovnih pravil s pomočjo sistema BRMS. Za implementacijo smo uporabili orodje Drools ter aplikacijski strežnik Wildfly 9. Obe tehnologiji sta odprtokodni ter podprti s strani podjetja Red Hat. Sistem Drools je eden izmed boljših odprtokodnih produktov BRMS, saj ponuja polno paleto orodij za zapis, shranjevanje ter izvajanje poslovnih pravil. Ponujeni vmesniki so primerni za tehnične in poslovne uporabnike. S tem je sistem Drools primeren za uporabo tudi v večjih poslovnih okoljih. Aplikacijski strežnik Wildfly se je v dosedanjih projektih izkazal kot hiter in zanesljiv produkt. To je, poleg dejstva, da so moduli sistema Drools predpripravljeni za uporabo na aplikacijskem strežniku Wildfly, tudi eden izmed ključnih razlogov za njegovo uporabo.

Implementirana poslovna pravila smo vzeli iz domene spletne trgovine. Prave spletne trgovine zagotovo vsebujejo še veliko več poslovnih pravil. V svoji implementaciji smo realizirali le najpomembnejša in najbolj očitna poslovna pravila glede na dosedanje izkušnje z uporabo spletnih trgovin. Pravila smo glede na njihovo naravo implementirali s pomočjo vodljivega uporabniškega vmesnika v obliki ločenih pravil oz. odločitvenih tabel. Z uporabo omenjenih načinov zapisa so pravila najbolj pregledna in strukturirana.

5.1 Uporabljeni moduli

Za implementacijo rešitve smo uporabili modula Drools Workbench in strežnik Kie, ki smo ju namestili na aplikacijski strežnik Wildfly. Ta mora biti zagnan v polnem načinu (angl. Full Profile), saj tako podpira vse tehnologije JEE 7, ki jih za svoje izvajanje potrebujeta izbrana modula.

V svoji implementaciji smo strežnik Kie zagnali v nadzorovanem načinu. Njegov upravljavec je modul Drools Workbench. Povezavo smo ustvarili s pomočjo naslednjih sistemskih lastnosti (angl. System Properties) aplikacijskega strežnika Wildfly:

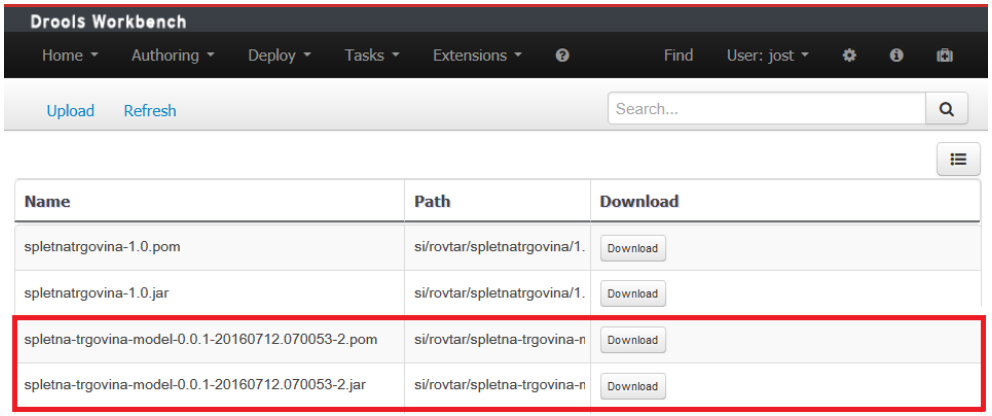
- **org.kie.server.id**: nastavlja enoličen identifikator strežnika Kie. Preko njega bo strežnik Kie identificiran pri upravljavcu. Omenjeni lastnosti smo nastavili vrednost *serverone*.
- **org.kie.server.location**: nastavlja lokacijo URL instance strežnika Kie, ki jo uporablja upravljavec za komunikacijo. Lastnosti smo nastavili vrednost *http://127.0.0.1:8081/kie-server/services/rest/server*.
- **org.kie.server.controller**: nastavlja lokacijo URL vmesnika REST upravljavca. Lastnosti smo nastavili vrednost *http://127.0.0.1:8081/kie-wb/rest/controller*.
- **org.kie.server.controller.user**: nastavlja uporabniško ime uporabnika, s katerim je upravljavec priključen na vmesnik REST strežnika Kie. Omenjeni lastnosti smo nastavili vrednost *kieserver*.
- **org.kie.server.controller.pwd**: nastavlja geslo uporabnika, s katerim je upravljavec priključen na vmesnik REST strežnika Kie. Lastnosti smo nastavili vrednost *kieserver1!*.

Dostop do strežnika Kie z zgoraj navedenim uporabniškim imenom in geslom omogočimo tako, da na aplikacijskem strežniku Wildfly ustvarimo aplikacijskega uporabnika (angl. Application User) z navedenimi podatki ter mu določimo vlogo *kie-server*.

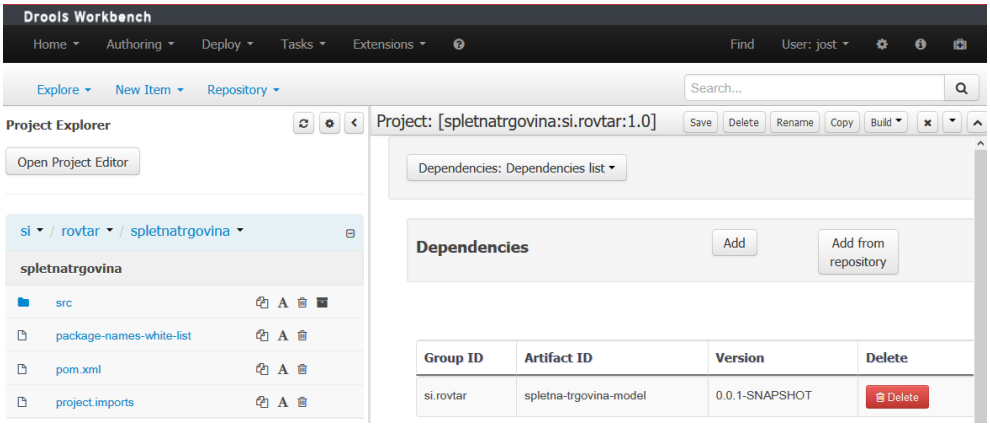
5.2 Priprava na implementacijo

Preden smo ustvarili projekt Kie (angl. Kie Project), v katerem smo implementirali poslovna pravila, smo znotraj orodja Drools Workbench ustvariti organizacijsko enoto (angl. Organizational Unit) in v njej repozitorij za beleženje različic Git. Oblikovanje različnih organizacijskih enot in repozitorijev Git omogoča uporabnikom orodja Drools Workbench ločevanje poslovnih pravil na različne sklope projektov oz. poslovne domene. V repozitoriju za beleženje različic Git nato ustvarimo projekt Kie.

Preden smo začeli z implementacijo poslovnih pravil, smo zgradili podatkovni model javanskih razredov, katere smo uvozili v Drools Workbench. Uvoz smo izvedli preko integriranega repozitorija artefaktov Maven. Uvožen artefakt vidimo na sliki 5.1. Za uporabo javanskih razredov uvoženega artefakta je potrebno v projektu s poslovnimi pravili nastaviti še odvisnost (angl. Dependency) na uvoženi artefakt. Nastavljeno odvisnost prikazuje slika 5.2. Podatkovne tipe lahko kreiramo tudi znotraj orodja Drools Workbench. Zunanja implementacija nam omogoča uporabo istih javanskih razredov v projektu s poslovnimi pravili in v aplikaciji programskega jezika Java, ki uporablja poslovna pravila.



Slika 5.1: Uvožen artefakt z implementacijo podatkovnega modela v obliki javanskih razredov.



Slika 5.2: Dodana odvisnost na artefakt z implementiranimi javanskimi razredi.

5.3 Implementacija poslovnih pravil

Vsak sistem BRMS ima drugačno sintakso zapisa poslovnih pravil, ki ga razume izvajalno okolje. V sistemu Drools se sintaksa imenuje *Drools Rule Language*. Po njem je tudi poimenovana končnica datotek (*drl*) v katerih so zapisana poslovna pravila. Splošna struktura poslovnega pravila je:

```
rule "ime poslovnega pravila"
        seznam atributov
when
        seznam pogojev
then
        seznam akcij
end
```

Ime poslovnega pravila mora biti enolično znotraj paketa pravil (angl. Rule Package). Atributi so neobvezni in določajo obnašanje posameznega pravila ali skupine pravil. Omogočajo nastavljanje vpliva na ostala pravila, dialekta, datumskega okvira veljavnosti pravila itd.

Navedeni pogoji preverjajo obstoj objekta določenega podatkovnega tipa in vrednost njegovih atributov. V primeru izpolnitve vseh navedenih pogojev se

izvedejo vse definirane akcije poslovnega pravila. V posameznem pogoju lahko definiramo tudi lokalne spremenljivke, ki so na voljo za uporabo v kasnejših pogojih in akcijah istega poslovnega pravila.

Za vsako poslovno pravilo oz. sklop poslovnih pravil smo ustvarili pakete (angl. Package) in jih tako vsebinsko ločili.

Poslovna pravila smo implementirali na dva načina: kot samostojna pravila ali kot odločitveno tabelo. V obeh primerih smo poslovna pravila implementirali preko vodljivega uporabniškega vmesnika, ki ga nudi orodje Drools Workbench. Implementacijo poslovnih pravil predstavimo v naslednjih poglavjih.

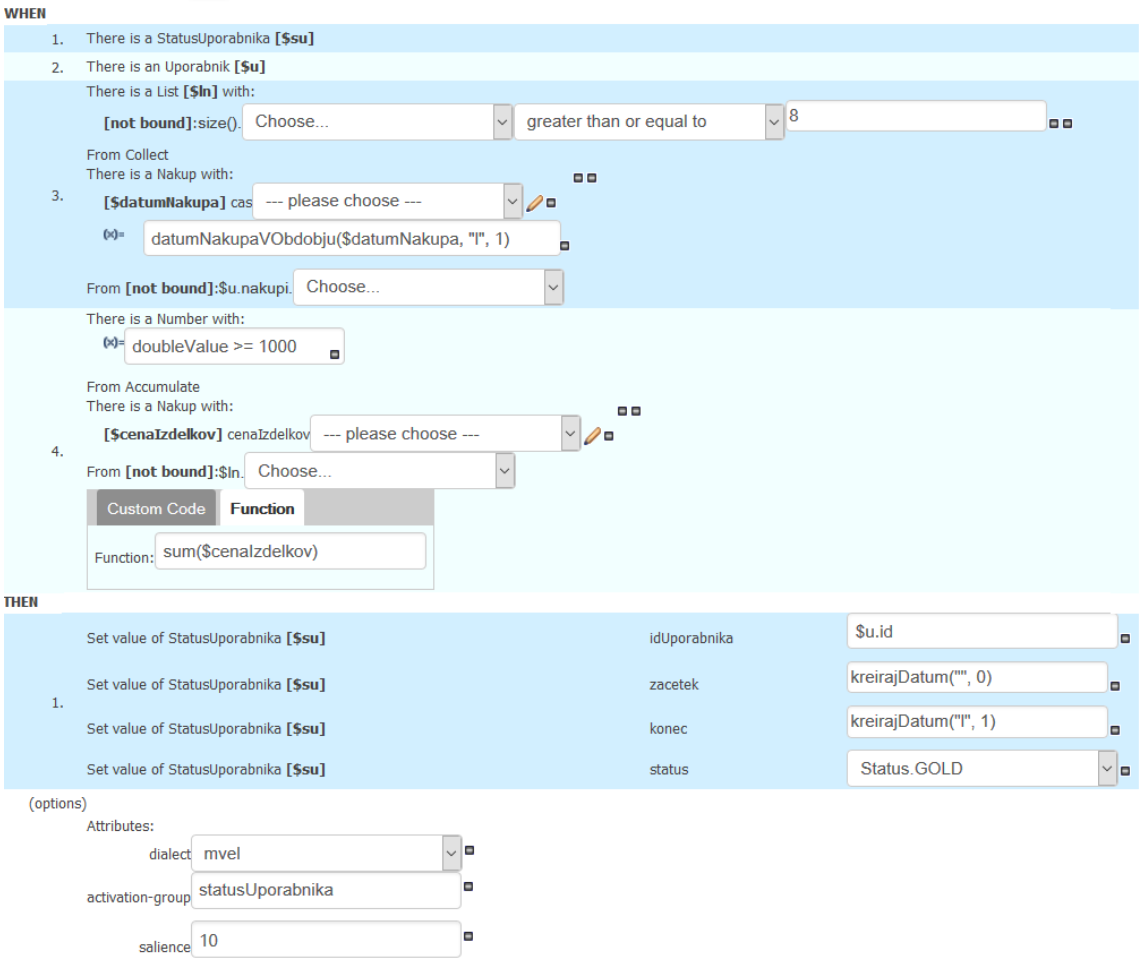
5.3.1 Poslovna pravila za določanje statusa uporabnika

Prvi sklop poslovnih pravil se nanaša na pridobivanje statusa uporabnika. Uporabnik si lahko glede na število in znesek nakupov v določenem časovnem obdobju pridobi status *silver* ali *gold*. Status je veljaven toliko časa, kot ga določa poslovno pravilo. Po preteku statusa ima uporabnik status *navaden*. Predvidoma se določanje statusa uporabnika izvede po vsakem njegovem nakupu. Poslovna pravila so prikazana v tabeli 5.1.

	Pogoji			Akcije	
#	**Število nakupov**	**Skupna vsota [€]**	**Časovno obdobje**	**Dodeljen status**	**Trajanje dodeljenega statusa**
1	>= 8	>= 1000	1 leto	GOLD	1 leto
2	>= 4	>= 600	6 mesecev	GOLD	6 mesecev
3	>= 6	>= 600	1 leto	SILVER	1 leto
4	>= 3	>= 300	6 mesecev	SILVER	6 mesecev

Tabela 5.1: Poslovna pravila za določanje statusa uporabnika.

Vsako pravilo smo implementirali v svoji datoteki. Implementacija prvega pravila je prikazana na sliki 5.3. Za izvajanje se mora vizualni zapis poslovnega pravila transformirati v splošni zapis, ki ga razume izvajalno okolje. Splošni zapis prvega pravila je prikazan na sliki 5.4.



Slika 5.3: Vizualna oblika poslovnega pravila za pridobivanje statusa *Gold* za obdobje enega leta.

```
rule "UporabnikPridobivanjeStatusaGoldLeto"
    dialect "mvel"
    activation-group "statusUporabnika"
    salience 10
    when
        $su : StatusUporabnika( )

        $u : Uporabnik( )

        $ln : List( size() >= 8 ) from collect (Nakup( $datumNakupa : cas,
                eval( datumNakupaVObdobju($datumNakupa, "l", 1) ))
                from $u.nakupi)
        Number( eval( doubleValue >= 1000 ))
            from accumulate (Nakup( $cenaIzdelkov : cenaIzdelkov) from $ln,
            sum($cenaIzdelkov))
    then
        $su.setIdUporabnika( $u.id );
        $su.setZacetek( kreirajDatum("", 0) );
        $su.setKonec( kreirajDatum("l", 1) );
        $su.setStatus( Status.GOLD );
end
```

Slika 5.4: Splošna oblika poslovnega pravila za pridobivanje statusa *Gold* za obdobje enega leta.

Prvi pogoj določa, da mora obstajati objekt tipa *StatusUporabnika*. Nanj se navezuje spremenljivka z imenom *$su*. Drugi pogoj določa, da mora obstajati objekt tipa *Uporabnik*. Nanj se navezuje spremenljivka z imenom *$u*. Tretji pogoj določa, da mora omenjeni objekt tipa *Uporabnik* vsebovati vsaj 8 nakupov, ki so bili opravljeni v zadnjem letu. Časovno ustreznost nakupa preverjamo s pomočjo metode *datumNakupaVObdobju*, ki smo jo ustvarili v ta namen. Njena implementacija je prikazana na sliki 5.5. Omenjena metoda vrača podatek tipa *boolean*, ki ga ovrednotimo s pomočjo vgrajene funkcije *eval*. Nakupe smo s pomočjo vgrajene funkcije *collect* zbrali v seznam tipa *List* in nanj vezali spremenljivko *$ln*. V četrtem pogoju smo s pomočjo vgrajene funkcije *accumulate* izpostavili atribut *cenaIzdelkov* vsakega od objektov tipa *Nakup* v seznamu, vezanega na spremenljivko $ln, in jih s pomočjo vgrajene funkcije *sum* sešteli. Seštevek smo priredili objektu tipa *Number* kot vrednost atributa *doubleValue*. Omenjeno vrednost na

koncu primerjamo s 1000 (skupna vsota vseh nakupov v definiranem časovnem obdobju mora biti minimalno 1000 eur), dobljeni izraz pa ovrednotimo s pomočjo vgrajene funkcije *eval*.

```
function Boolean datumNakupaVObdobju(Calendar datumNakupa, String obdobje,
             int st){

    if(st < 0 || datumNakupa == null){
        return false;
    }

    Calendar c = kreirajDatum(obdobje, -st);

    if(datumNakupa.compareTo(c) > 0){
        return true;
    }else{
        return false;
    }
}

function Calendar kreirajDatum(String obdobje, int st){

    Calendar c = Calendar.getInstance();
    c.set(Calendar.HOUR_OF_DAY, 0);
    c.set(Calendar.MINUTE, 0);
    c.set(Calendar.SECOND, 0);
    c.set(Calendar.MILLISECOND, 0);

    if("m".equals(obdobje)){
        c.add(Calendar.MONTH, st);
    }else if ("l".equals(obdobje)){
        c.add(Calendar.YEAR, st);
    }

    return c;
}
```

Slika 5.5: Metodi za določanje časovne ustreznosti nakupa.

Ob izpolnitvi vseh zgoraj navedenih pogojev se izvedejo štiri akcije, ki se vse navezujejo na objekt tipa *StatusUporabnika* preko spremenljivke *$su*. Vsaka akcija predstavlja klic ene funkcije objekta. S klici funkcij identificiramo uporabnika (ID

številka), določimo pridobljen status ter začetek in konec veljave pridobljenega statusa.

Poslovnim pravilom smo določili tudi tri atribute. Atribut z imenom *dialect* in vrednostjo *mvel* določa narečje jezika DRL, ki smo ga uporabili za zapis poslovnega pravila. Drugi atribut z imenom *activation-group* določa aktivacijsko skupino, kateri pripada poslovno pravilo. Ko se sproži pravilo iz aktivacijske skupine, se iz vrste za izvajanje umaknejo vsa ostala poslovna pravila, ki pripadajo isti aktivacijski skupini. To pomeni, da se lahko izvede le eno poslovno pravilo iz posamezne aktivacijske skupine. Zadnji atribut z imenom *salience* in vrednostjo *10* določa prioriteto poslovnega pravila pri razvrščanju v vrsto za izvajanje. Števila so lahko pozitivna ali negativna. Poslovno pravilo z višjo številko ima višjo prioriteto in se bo izvedlo prej kot pravilo z nižjo prioriteto.

Atributa *salience* in *activation-group* smo definirali, ker se pogoji nekaterih poslovnih pravil med seboj prekrivajo. Npr. če je uporabnik v obdobju zadnjega pol leta opravil šest nakupov v skupni vrednosti od 600,00 evra do 999,99 evra potem glede na tabelo 5.1 ustreza pravilom 2, 3 in 4. Z atributoma *salience* in *activation-group* določimo, katero poslovno pravilo ima višjo prioriteto, ter na ta način izvedemo le eno - poslovno pravilo št. 2.

Zgoraj smo opisali implementacijo prvega poslovnega pravila. Vsa ostala pravila imajo popolnoma enako zgradbo, razlikujejo se le v omejitvenih vrednostih pogojev in določitvijo statusa ter njegovega trajanja. Te vrednosti so razvidne iz tabele 5.1. Atributa *dialect* in *activation group* sta določena vsem pravilom z enakimi vrednostmi. Atribut *salience* je sicer določen vsem pravilom, vrednosti pa so različne. Poslovno pravilo, ki je v tabeli 5.1 višje, ima višjo vrednost atributa *salience* in s tem višjo prioriteto.

5.3.2 Poslovna pravila za določanje cene poštnine

Drugi sklop poslovnih pravil se nanaša na določanje cene poštnine nakupa. Cena poštnine se določi glede na status kupca, skupno ceno izdelkov nakupa, vrsto plačila in vrsto dostave. Implementacija poslovnih pravil z uporabo odločitvenih tabel v orodju Drools Workbench je prikazana na sliki 5.6. Vsaka vrstica v odločitveni tabeli predstavlja eno poslovno pravilo. Splošen zapis drugega poslovnega pravila v odločitveni tabeli je prikazan na sliki 5.7.

#	Desc	status kupca	cena izdelkov od [€]	cena izdelkov do [€]	vrsta plačila	vrsta dostave	cena postnine [€]
		NakupRE [$nakup]					$nakup
		statusKupca [==]	cenaIzdelkov [>=]	cenaIzdelkov [<]	vrstaPlacila [==]	vrstaDostave [==]	cenaPostnine
1		Status.NAVADEN	500				0
2		Status.NAVADEN	300	500	VrstaPlacila.PO_PREVZETJU	VrstaDostave.PBS	1
3		Status.NAVADEN	300	500	VrstaPlacila.PO_PREVZETJU	VrstaDostave.HITRA_POSTA	2
4		Status.NAVADEN	300	500	VrstaPlacila.PREKO_SPLETA	VrstaDostave.PBS	0
5		Status.NAVADEN	300	500	VrstaPlacila.PREKO_SPLETA	VrstaDostave.HITRA_POSTA	1
6		Status.NAVADEN	300	500	VrstaPlacila.PO_OBROKIH	VrstaDostave.PBS	1
7		Status.NAVADEN	300	500	VrstaPlacila.PO_OBROKIH	VrstaDostave.HITRA_POSTA	2
8		Status.NAVADEN	300	500	VrstaPlacila.PREDRACUN	VrstaDostave.PBS	0
9		Status.NAVADEN	300	500	VrstaPlacila.PREDRACUN	VrstaDostave.HITRA_POSTA	1
10		Status.NAVADEN	0	300	VrstaPlacila.PO_PREVZETJU	VrstaDostave.PBS	2
11		Status.NAVADEN	0	300	VrstaPlacila.PO_PREVZETJU	VrstaDostave.HITRA_POSTA	3
12		Status.NAVADEN	0	300	VrstaPlacila.PREKO_SPLETA	VrstaDostave.PBS	1
13		Status.NAVADEN	0	300	VrstaPlacila.PREKO_SPLETA	VrstaDostave.HITRA_POSTA	2
14		Status.NAVADEN	0	300	VrstaPlacila.PO_OBROKIH	VrstaDostave.PBS	2
15		Status.NAVADEN	0	300	VrstaPlacila.PO_OBROKIH	VrstaDostave.HITRA_POSTA	3
16		Status.NAVADEN	0	300	VrstaPlacila.PREDRACUN	VrstaDostave.PBS	1
17		Status.NAVADEN	0	300	VrstaPlacila.PREDRACUN	VrstaDostave.HITRA_POSTA	2
18		Status.SILVER	200				0
19		Status.SILVER	0	200	VrstaPlacila.PO_PREVZETJU	VrstaDostave.PBS	2
20		Status.SILVER	0	200	VrstaPlacila.PO_PREVZETJU	VrstaDostave.HITRA_POSTA	3
21		Status.SILVER	0	200	VrstaPlacila.PREKO_SPLETA	VrstaDostave.PBS	1
22		Status.SILVER	0	200	VrstaPlacila.PREKO_SPLETA	VrstaDostave.HITRA_POSTA	2
23		Status.SILVER	0	200	VrstaPlacila.PO_OBROKIH	VrstaDostave.PBS	2
24		Status.SILVER	0	200	VrstaPlacila.PO_OBROKIH	VrstaDostave.HITRA_POSTA	3
25		Status.SILVER	0	200	VrstaPlacila.PREDRACUN	VrstaDostave.PBS	1
26		Status.SILVER	0	200	VrstaPlacila.PREDRACUN	VrstaDostave.HITRA_POSTA	2
27		Status.GOLD					0

Slika 5.6: Poslovna pravila za določanje cene poštnine nakupa.

```
rule "Row 2 DolocanjeCenePostnine"
    dialect "mvel"
    when
        $nakup : NakupRE( statusKupca == Status.NAVADEN ,
                          cenaIzdelkov >= 300.0 , cenaIzdelkov < 500.0 ,
                          vrstaPlacila == VrstaPlacila.PO_PREVZETJU ,
                          vrstaDostave == VrstaDostave.PBS )
    then
        $nakup.setCenaPostnine( 1 );
end
```

Slika 5.7: Splošna predstavitev poslovnega pravila za določanje cene poštnine.

Posamezno poslovno pravilo vsebuje le pogoj, ki preverja obstoj objekta tipa *NakupRE* z določenimi vrednostmi njegovih atributov. Ta se razlikuje od tipa *Nakup* in je namenjen uporabi le v poslovnih pravilih. To označujeta zadnji dve črki imena tipa *RE*, ki pomenita *rule engine*. Nov tip smo definirali z namenom, da v poslovna pravila posredujemo le podatke, ki se uporabljajo v poslovnih pravilih. Tako dosežemo krajši čas prenosa podatkov od poslovne aplikacije do izvajalnega okolja poslovnih pravil. Posamezno poslovno pravilo ima definirano akcijo, ki nastavlja vrednost atributa *cenaPostnine* obravnavanega objekta tipa *NakupRE*.

5.3.3 Poslovna pravila za določanje vrste in vrednosti darilnega bona

Tretji sklop poslovnih pravil se nanaša na določanje vrste in vrednosti darilnega bona, ki se ob novem letu generirajo za vse uporabnike za preteklo koledarsko leto. Darilni bon določa popust ali vsoto pri nakupu enega izdelka iz določene kategorije in se ustvari v primeru, da je uporabnik v preteklem letu kupil izdelke iz določene kategorije z minimalno skupno vrednostjo, ki jo določa poslovno pravilo. Odločitveno tabelo z definiranimi poslovnimi pravili prikazuje slika 5.8. Splošni zapis prvega pravila iz slike 5.8 prikazuje slika 5.9.

#	Description	znesek nakupov od	znesek nakupov do	vrsta bona	vrednost bona
		DarilniBonRE [$d]		$d	$d
		znesekNakupov [>=]	znesekNakupov [<]	vrstaBona	vrednost
1		0	100	VrstaDarilnegaBona.ODSTOTKI	0.0
2		100	200	VrstaDarilnegaBona.ODSTOTKI	0.05
3		200	300	VrstaDarilnegaBona.ODSTOTKI	0.1
4		300	500	VrstaDarilnegaBona.ODSTOTKI	0.15
5		500	1000	VrstaDarilnegaBona.ODSTOTKI	0.2
6		1000		VrstaDarilnegaBona.ODSTOTKI	0.25

Slika 5.8: Poslovna pravila za določanje vrednosti in vrste darilnega bona za posamezno kategorijo.

```
rule "Row 1 GeneriranjeBonaKonecLeta"
    dialect "mvel"
    when
        $d : DarilniBonRE( znesekNakupov >= 0 , znesekNakupov < 100 )
    then
        $d.setVrstaBona( VrstaDarilnegaBona.ODSTOTKI );
        $d.setVrednost( 0.0 );
end
```

Slika 5.9: Splošna predstavitev poslovnih pravil za generiranje darilnega bona.

Posamezno pravilo vsebuje pogoj, ki preverja obstoj objekta tipa *DarilniBonRE* in pravilno vrednost njegovega atributa *znesekNakupov*. Enako kot *NakupRE* je bil tudi tip *DarilniBonRE* ustvarjen le za uporabo v poslovnih pravilih. Definirani akciji pravila določata, da se obravnavanemu objektu glede na vrednost atributa *vrednostNakupa* določi vrsto in vrednost bona. Trenutna implementacija sklopa pravil za vse kategorije določa enake omejitvene zneske za določanje bona. Možna nadgradnja bi bila dopolnitev pogojev s preverjanjem identifikatorja kategorije. Tako bi lahko za vsako kategorijo posebej določali omejitvene zneske in vrednosti bona.

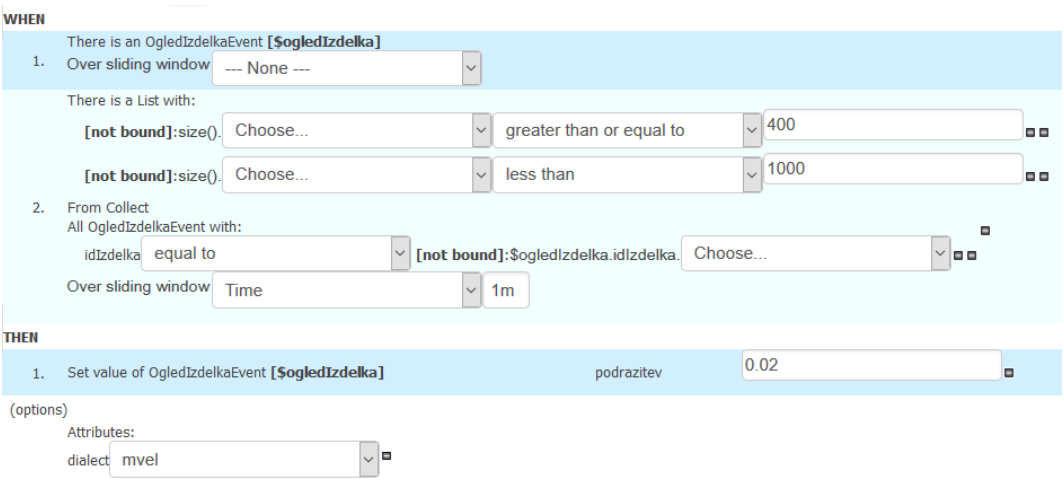
5.3.4 Poslovna pravila za določanje podražitve izdelka

Zadnji sklop poslovnih pravil se nanaša na dviganje cene izdelka glede na število ogledov v definiranem časovnem okviru. Poslovna pravila so predstavljena v preglednici 5.2.

	Pogoji		Akcije
#	**Število ogledov**	**Časovni okvir [min]**	**Podražitev [%]**
1	0—399	1	0
2	400—999	1	2
3	>= 1000	1	5

Tabela 5.2: Poslovna pravila za določanje podražitve izdelka glede na število ogledov v zadnji minuti.

Za implementacijo tega sklopa pravil smo poleg modula Drools Expert uporabili tudi njegovo razširitev Drools Fusion, ki omogoča procesiranje dogodkov. Uporabili smo funkcijo časovnega drsečega okna (angl. Sliding Time Wildows), ki nam omogoča ovrednotenje poslovnih pravil iz podatkov oz. dogodkov, ki so bili dodani v sejo v zadnjih X časovnih enotah. Implementacijo poslovnega pravila, ki je v preglednici drugi po vrsti prikazuje slika 5.10. Splošen zapis tega pravila pa je predstavljen na sliki 5.11.



Slika 5.10: Poslovno pravilo za določanje dviga cene izdelka glede na število ogledov v zadnji 1 minuti.

```
rule "OgledIzdelka2"
    dialect "mvel"
    when
        $ogledIzdelka : OgledIzdelkaEvent( )
        List( size() >= 400 , size() < 1000 )
             from collect ( OgledIzdelkaEvent( idIzdelka == $ogledIzdelka.idIzdelka )
             over window:time (1m))
    then
        $ogledIzdelka.setPodrazitev( 0.02 );
end
```

Slika 5.11: Splošna predstavitev poslovnih pravil za določanje dviga cene izdelka.

Da lahko na objektu tipa *OgledIzdelkaEvent* uporabljamo funkcije, ki jih nudi razširitev Drools Fusion, ga moramo označiti kot dogodek (angl. Event). Deklaracijo prikazuje slika 5.12, zapisali pa smo jo v ločeni datoteki v istem paketu kot poslovna pravila.

```
declare OgledIzdelkaEvent
    @role (event)
end
```

Slika 5.12: Tipu *OgledIzdelkaEvent* določena vloga *dogodek*.

Prvi pogoj pravila preverja obstoj objekta tipa *OgledIzdelkaEvent*. Drugi pogoj je sestavljen iz dveh delov, kjer prvi del (*collect (OgledIzdelkaEvent(idIzdelka == $ogledIzdelka.idIzdelka) over window:time (1m))*) zbere vse objekte tipa *OgledIzdelkaEvent*, ki imajo enako vrednost atributa *idIzdelka* in so bili v sejo dodani pred največ eno minuto ter nato doda v seznam - objekt tipa *List*. Drugi del (*List(size() >= 400 , size() <1000)*) določa obstoj med 400 in 1000 objektov, ki zadostujejo prvemu delu pogoja. Poslovno pravilo vsebuje akcijo, ki nastavlja vrednost atributa *podrazitev* obravnavanega objekta tipa *OgledIzdelkaEvent* na vrednost *0,02*.

5.3.5 Kreiranje zbirk Kie in sej Kie

Kot smo zapisali na začetku poglavja, se poslovna pravila vedno izvajajo znotraj seje. Za njihovo instanciranje in upravljanje je odgovoren vsebnik Kie. V ustvarjenem projektu Drools smo vsak sklop poslovnih pravil implementirali v svojem paketu. Iz vsakega paketa smo eksplicitno definirali zbirko Kie in iz vsake zbirke sejo Kie. Deklaracijo zbirk in sej Kie znotraj projekta Drools prikazuje slika 5.13.

```
<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <kbase name="kbaseOgledIzdelka" default="false"
           eventProcessingMode="stream" equalsBehavior="identity"
           packages="si.rovtar.spletnatrgovina.poslovnapravila.ogledizdelka">
    <ksession name="ogledizdelka" type="stateful" default="false" clockType="realtime"/>
  </kbase>
  <kbase name="kbaseCenaPostnine" default="false"
           eventProcessingMode="stream" equalsBehavior="identity"
           packages="si.rovtar.spletnatrgovina.poslovnapravila.cenapostnine">
    <ksession name="cenaPostnine" type="stateless" default="false" clockType="realtime"/>
  </kbase>
  <kbase name="kbaseStatusUporabnika" default="false"
           eventProcessingMode="stream" equalsBehavior="identity"
           packages="si.rovtar.spletnatrgovina.poslovnapravila.statusuporabnika">
    <ksession name="pridobiStatusUporabnika" type="stateless" default="false"
              clockType="realtime"/>
  </kbase>
  <kbase name="kbaseGeneriranjeBona" default="false"
           eventProcessingMode="stream" equalsBehavior="identity"
           packages="si.rovtar.spletnatrgovina.poslovnapravila.generiranjebona">
    <ksession name="generiranjeBona" type="stateless" default="false" clockType="realtime"/>
  </kbase>
</kmodule>
```

Slika 5.13: Deklaracija zbirk Kie in sej Kie v datoteki kmodule.xml.

Za tako implementacijo smo se odločili, ker se v primeru razširitve in uporabe istih tipov objektov v različnih sklopih poslovnih pravil lahko zgodi, da pride do nepredvidenega izvajanja in ovrednotenja poslovnih pravil ter posledično do napačnih rezultatov. Če se v prihodnosti pokaže potreba po združevanju sklopov poslovnih pravil, projekt Drools omogoča ustvarjanje sej Kie iz več zbirk Kie.

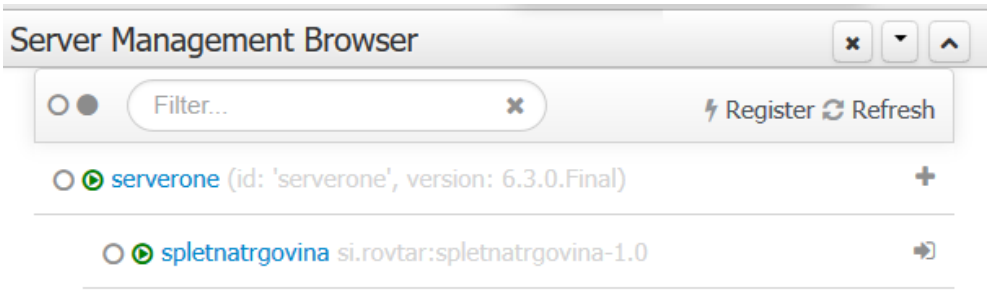
Prva seja Kie z imenom *ogledizdelka* med posameznim ovrednotenjem poslovnih pravil shranjuje stanje. To zahteva uporaba funkcij razširitve Drools Fusion. Zadnje tri seje Kie ne shranjujejo stanja med posameznimi ovrednotenji, saj želimo, da se vsako izvrši nad novim sklopom podatkov, ki ga pred vsakim ovrednotenjem dodamo v sejo.

5.3.6 Namestitev in izvajanje poslovnih pravil

Po končani implementaciji je za uporabo poslovnih pravil potrebna njihova namestitev v izvajalno okolje - na strežnik Kie. Ker smo orodje Drools Workbench integrirali s strežnikom Kie je namestitev hitra in enostavna.

Ob kliku na gumb *Buid & deploy* v nastavitvah projekta Drools se ta prevede, zapakira v arhiv JAR in odloži v integrirani repozitorij Maven.

V elementu menija *Deploy* izberemo *Rule deployments*. Odpre se nam stran, na kateri lahko upravljamo povezane strežnike Kie. Ker smo strežnik Kie zagnali v nadzorovanem načinu in za upravljavca določili orodje Kie Workbench, se je na seznamu pokazal strežnik Kie z definiranim imenom *serverone*. Znotraj njega smo iz ustvarjenega projekta Drools instancirali vsebnik Kie z imenom *spletnatrgovina*. Povezan strežnik Kie in instanciran vsebnik Kie je prikazan na sliki 5.14.



Slika 5.14: Povezan strežnik Kie in instanciran vsebnik Kie.

V primeru spreminjanja poslovnih pravil v projektu Drools je potrebna še njihova posodobitev v izvajalnem okolju. To lahko storimo s klikom na gumb *Upgrade* v podrobnostih vmesnika Kie. Druga možnost je vzpostavitev bralnika (angl. Scanner), ki na definiran časovni interval preverja repozitorij Maven za nove različice projekta Drools in samodejno posodobi vsebnik Kie z novo različico.

Poslovna pravila so z instanciranjem vsebnika Kie nameščena v izvajalno okolje in na voljo za uporabo v poslovnih aplikacijah. Naslov, kjer se nahaja instanciran vsebnik Kie, je določen po modelu: http:// ali https://[*naslov strežnika*] : [*vrata*] /kie-server/services/rest/server/containers/instances/[*ime vsebnika*]. Ker aplikacijski strežnik Wildfly teče na lokalnem računalniku, je naslov za dostop do instanciranega vsebnika:

http://127.0.0.1:8081/kie-server/services/rest/server/containers/instances/spletnatrgovina.

Strežnik Kie za odstop oz. ovrednotenje poslovnih pravil ponuja vmesnik REST ter vmesnik Java Client, ki zelo poenostavi dostop do poslovnih pravil iz aplikacij programskega jezika Java.

5.3.7 Ovrednotenje poslovnih pravil in diskusija

Glede na podatke in vzorec pogojev in akcij poslovnih pravil za določanje statusa uporabnika in podražitve izdelka bi bila uporaba odločitvene tabele na prvi pogled bolj smiselna. Ker pa uporaba funkcij *collect*, *accumulate* ter časovnega drsečega okna v odločitveni tabeli ni priporočljiva in ni podprta pri izgradnji tabele s pomočjo vodljivega uporabniškega vmesnika, smo se odločili implementirati vsako pravilo posebej.

Implementirana poslovna pravila za generiranje darilnega bona predvidevajo, da aplikacija sama poskrbi za seštevek zneskov izdelkov v posamezni kategoriji za vsakega uporabnika v zadnjem letu. Druga možnost bi bila, da bi v poslovna pravila dodali tudi seštevanje zneskov nakupov. Tako bi pridobili veliko možnosti za razširjanje poslovnih pravil v prihodnosti. Za primerjavo delovanja smo realizirali tudi drugo možnost. Implementacijo enega izmed pravil prikazuje slika 5.15. Na njej lahko vidimo, da se kompleksnost poslovnega pravila precej poveča.

Primerjali smo tudi čase izvajanja obeh implementacij pravila. Testni podatki so obsegali 11 kategorij in enega uporabnika z 10 nakupi, kjer vsak vsebuje po 3 izdelke. V prvi implementaciji pravil smo na strani aplikacije za vsako kategorijo sešteli zneske izdelkov ter iz njih kreirali objekte tipa *DarilniBonRE*, ki smo jih dodali zahtevi za ovrednotenje poslovnih pravil. V drugi implementaciji smo zahtevi za ovrednotenje pravil dodali celoten objekt tipa *Uporabnik* z vsemi nakupi in izdelki, ker je izbor primernih izdelkov po kategorijah izveden v poslovnih pravilih. Pri obeh različicah smo izmerili čas seštevanja zneskov izdelkov po posamezni kategoriji in določanje vrste ter vrednosti darilnega bona. V povprečju je bila originalna (prva) implementacija vsaj 1,5-krat hitrejša. Čas trajanja pri izvedbi s seštevanjem zneskov izdelkov na strani aplikacije je 13ms, čas trajanja pri izvedbi s seštevanjem zneskov v poslovnih pravilih je 20ms. S povečevanjem števila uporabnikov in njihovih nakupov, ki smo jih naenkrat podali za ovrednote-

```
rule "darilni bon 1"
     no-loop true
when
    $kategorija : KategorijaIzdelka( )
    $u : Uporabnik( $n: nakupi )

    $nakupi : ArrayList()
        from collect(Nakup($casNakupa : cas,
                datumVObdobju($casNakupa, "l", 1)) from $n)
    $izdelki : ArrayList(size > 0)
        from accumulate  (Nakup($izdelki2 : izdelki)
                from $nakupi, init(List l = new ArrayList(); )
                action(l.addAll($izdelki2); ) result (l) )
    $izdelkiKategorije : ArrayList(size > 0)
        from collect(Izdelek(kategorija.id == $kategorija.id)
                from $izdelki )
    accumulate(Izdelek($cenaIzdelka : cena)
        from $izdelkiKategorije; $sestevek : sum($cenaIzdelka);
                $sestevek >= 100 && < 200)
then
    DarilniBonRE db = new DarilniBonRE($u.getId(), $kategorija.getId(),
        $sestevek.doubleValue(), VrstaDarilnegaBona.ODSTOTKI, 0.05);
    listPridobljeniBoni.add(db);
end
```

Slika 5.15: Implementacija poslovnih pravil za generiranje vrste in vrednosti darilnega bona.

nje v poslovna pravila se je razlika le še povečevala. Velikost podane podatkovne entitete v notaciji XML (podatkovna entiteta se pri prenosu iz testne aplikacije v strežnik Kie serializira v notacijo XML) pri zgoraj omenjenem testnem primeru je bila pri prvi implementaciji 3600 bajtov, pri drugi pa 20400 bajtov (5,6-krat več). Iz tega je razvidno, da je razlog za počasnejše izvajanje celotnega procesa (izbor izdelkov, klic strežnika Kie in določanje darilnega bona) veliko večja količina prenesenih podatkov na strežnik Kie in ne počasnejše izvajanje poslovnih pravil.

Orodje Drools Workbench za namen testiranja poslovnih pravil ponuja možnost definiranja testnih scenarijev. V posameznem scenariju definiramo vhodne podatke in pričakovane izhodne podatke ter zbirko znanja in sejo, v katero se definirani podatki vstavijo. Ker lahko v testnih scenarijih izberemo le seje, ki med ovrednotenji pomnijo stanje, naša implementacija pa definira le eno tako sejo, smo se odločili, da teste kreiramo v ločenem modulu projekta Maven, v katerem smo ustvarili javanske razrede objektnega modela uporabljene v poslovnih pravilih. Teste smo kreirali s

pomočjo ogrodja JUnit v javanskem razredu *PoslovnaPravilaTest.java*. Ta vsebuje 4 metode (za vsako skupino poslovnih pravil) s testnimi primeri. Izbrani testni primeri preverjajo vse mejne vrednosti, pri katerih je posamezno poslovno pravilo pozitivno. Na ta način smo kreirali popoln nabor testnih primerov za implementirana poslovna pravila. Implementirali smo tudi pomožne razrede za kreiranje potrebnih objektov poslovnega modela ter klicanje vmesnika REST strežnika Kie.

Prva metoda *testPridobivanjeStatusa* vsebuje 17 testnih primerov za pridobivanje statusa uporabnika. Implementacijo metode prikazuje slika 5.18. Test je sestavljen tako, da najprej ustvarimo dva seznama. V prvega dodajamo objekte tipa *Uporabnik*, pri katerih definiramo nakupe z zneski določenih vrednosti. Uporabnika in pripadajoče nakupe z zneski kreiramo s pomočjo metode *kreirajUporabnikaTestPridobiStatus*, ki se nahaja v razredu *ModelHelper*. V drug seznam dodajamo objekte tipa *StatusUporabnika*, ki predstavljajo pričakovane vrednosti po ovrednotenju podatkov v poslovnih pravilih. Po kreiranju vseh testnih podatkov in pričakovanih rezultatov v zanki pokličemo poslovna pravila za vsakega definiranega uporabnika. Klic poslovnih pravil izvedemo s klicem funkcije *pridobiStatusUporabnika*. Po prejemu rezultatov ovrednotenja poslovnih pravil primerjamo pričakovan status in trajanje ter dejanski status in trajanje, ki jih pridobimo iz poslovnih pravil. Primerjavo izvedemo s pomočjo funkcije *assertEquals*, ki je del ogrodja JUnit.

```
@Test
public void testPridobivanjeStatusa() {

    List<Uporabnik> lu = new ArrayList<Uporabnik>();
    List<StatusUporabnika> ls = new ArrayList<StatusUporabnika>();

    lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(3, 299,
                        ModelHelper.kreirajDatum("m", -6, false)));
    ls.add(new StatusUporabnika(Status.NAVADEN,
            ModelHelper.kreirajDatum("", 0, true),
            ModelHelper.kreirajDatum("", 0, true), 123));

    lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(2, 300,
            ModelHelper.kreirajDatum("m", -6, false)));
    ls.add(new StatusUporabnika(Status.NAVADEN,
            ModelHelper.kreirajDatum("", 0, true),
            ModelHelper.kreirajDatum("", 0, true), 123));

    lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(3, 300,
            ModelHelper.kreirajDatumRazlika("m", -6, -1)));
    ls.add(new StatusUporabnika(Status.NAVADEN,
            ModelHelper.kreirajDatum("", 0, true),
            ModelHelper.kreirajDatum("m", 6, true), 123));

    lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(3, 300,
            ModelHelper.kreirajDatum("m", -6, false)));
    ls.add(new StatusUporabnika(Status.SILVER,
            ModelHelper.kreirajDatum("", 0, true),
            ModelHelper.kreirajDatum("m", 6, true), 123));

    lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(3, 599,
            ModelHelper.kreirajDatum("m", -6, false)));
    ls.add(new StatusUporabnika(Status.SILVER,
            ModelHelper.kreirajDatum("", 0, true),
            ModelHelper.kreirajDatum("m", 6, true), 123));

    lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(4, 599,
            ModelHelper.kreirajDatum("m", -6, false)));
    ls.add(new StatusUporabnika(Status.SILVER,
            ModelHelper.kreirajDatum("", 0, true),
            ModelHelper.kreirajDatum("m", 6, true), 123));

    lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(5, 599,
            ModelHelper.kreirajDatum("m", -6, false)));
    ls.add(new StatusUporabnika(Status.SILVER,
            ModelHelper.kreirajDatum("", 0, true),
            ModelHelper.kreirajDatum("m", 6, true), 123));
```

Slika 5.16: Izvorna koda testov za pridobivanje statusa uporabnika - 1. del.

```
lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(6, 600,
        ModelHelper.kreirajDatumRazlika("m", -6, -1)));
ls.add(new StatusUporabnika(Status.SILVER,
        ModelHelper.kreirajDatum("", 0, true),
        ModelHelper.kreirajDatum("l", 1, true), 123));

lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(6, 600,
        ModelHelper.kreirajDatum("l", -1, false)));
ls.add(new StatusUporabnika(Status.SILVER,
        ModelHelper.kreirajDatum("", 0, true),
        ModelHelper.kreirajDatum("l", 1, true), 123));

lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(4, 1000,
        ModelHelper.kreirajDatum("l", -1, false)));
ls.add(new StatusUporabnika(Status.SILVER,
        ModelHelper.kreirajDatum("", 0, true),
        ModelHelper.kreirajDatum("m", 6, true), 123));

lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(5, 1000,
        ModelHelper.kreirajDatum("l", -1, false)));
ls.add(new StatusUporabnika(Status.SILVER,
        ModelHelper.kreirajDatum("", 0, true),
        ModelHelper.kreirajDatum("m", 6, true), 123));

lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(4, 600,
        ModelHelper.kreirajDatum("m", -6, false)));
ls.add(new StatusUporabnika(Status.GOLD,
        ModelHelper.kreirajDatum("", 0, true),
        ModelHelper.kreirajDatum("m", 6, true), 123));

lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(5, 600,
        ModelHelper.kreirajDatum("m", -6, false)));
ls.add(new StatusUporabnika(Status.GOLD,
        ModelHelper.kreirajDatum("", 0, true),
        ModelHelper.kreirajDatum("m", 6, true), 123));

lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(6, 1000,
        ModelHelper.kreirajDatum("m", -6, false)));
ls.add(new StatusUporabnika(Status.GOLD,
        ModelHelper.kreirajDatum("", 0, true),
        ModelHelper.kreirajDatum("m", 6, true), 123));

lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(8, 999,
        ModelHelper.kreirajDatum("l", -1, false)));
ls.add(new StatusUporabnika(Status.GOLD,
        ModelHelper.kreirajDatum("", 0, true),
        ModelHelper.kreirajDatum("m", 6, true), 123));
```

Slika 5.17: Izvorna koda testov za pridobivanje statusa uporabnika - 2. del.

```
        lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(8, 1000,
                ModelHelper.kreirajDatumRazlika("l", -1, -1)));
        ls.add(new StatusUporabnika(Status.GOLD,
                ModelHelper.kreirajDatum("", 0, true),
                ModelHelper.kreirajDatum("m", 6, true), 123));

        lu.add(ModelHelper.kreirajUporabnikaTestPridobiStatus(8, 1000,
                ModelHelper.kreirajDatum("l", -1, false)));
        ls.add(new StatusUporabnika(Status.GOLD,
                ModelHelper.kreirajDatum("", 0, true),
                ModelHelper.kreirajDatum("l", 1, true), 123));

        for (int j = 0; j < lu.size(); j++) {

            Uporabnik up = lu.get(j);
            StatusUporabnika pricakovaniStatus = ls.get(j);

            try {
                StatusUporabnika dejanskiStatus =
                        pp.pridobiStatusUporabnika(up, rulesClient);

                assertEquals(pricakovaniStatus.getIdUporabnika(),
                        dejanskiStatus.getIdUporabnika());
                assertEquals(pricakovaniStatus.getStatus(),
                        dejanskiStatus.getStatus());

                if (dejanskiStatus.getStatus() != Status.NAVADEN) {
                    assertEquals(pricakovaniStatus.getZacetek(),
                            dejanskiStatus.getZacetek());
                    assertEquals(pricakovaniStatus.getKonec(),
                            dejanskiStatus.getKonec());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

Slika 5.18: Izvorna koda testov za pridobivanje statusa uporabnika - 3. del.

Druga metoda *testDolocanjeCenePostnine* vsebuje 72 primerov za testiranje določanja cene poštnine nakupa. Njena implementacija je prikazana na sliki 5.26. Podobno kot prva metoda tudi druga na začetku definira dva seznama. V prvega dodajamo objekte tipa *NakupRE*, ki ga kreiramo s pomočjo metode *kreirajNakupRE*. V drug seznam dodajamo objekte tipa *Double*, ki predstavljajo pričakovan rezultat iz poslovnih pravil. Po kreiranju vseh testnih podatkov za vsak ustvarjeni nakup pokličemo poslovna pravila ter primerjamo dobljen rezultat s pričakovanim. Klic poslovnih pravil se izvede s klicem funkcije *pridobiCenoPostineNakupa*, primerjava vrednosti pa s klicem funkcije *assertEquls*.

```
@Test
public void testDolocanjeCenePostnine() {

    List<NakupRE> seznamNakupov = new ArrayList<NakupRE>();
    List<Double> seznamPricakovanihCenPostnin = new ArrayList<Double>();

    seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 500,
            VrstaDostave.PBS, VrstaPlacila.PO_OBROKIH));
    seznamPricakovanihCenPostnin.add(Double.valueOf(0));

    seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 500,
            VrstaDostave.PBS, VrstaPlacila.PO_PREVZETJU));
    seznamPricakovanihCenPostnin.add(Double.valueOf(0));

    seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 500,
            VrstaDostave.PBS, VrstaPlacila.PREDRACUN));
    seznamPricakovanihCenPostnin.add(Double.valueOf(0));

    seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 500,
            VrstaDostave.PBS, VrstaPlacila.PREKO_SPLETA));
    seznamPricakovanihCenPostnin.add(Double.valueOf(0));

    seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 500,
            VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_OBROKIH));
    seznamPricakovanihCenPostnin.add(Double.valueOf(0));

    seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 500,
            VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_PREVZETJU));
    seznamPricakovanihCenPostnin.add(Double.valueOf(0));
```

Slika 5.19: Izvorna koda testov za določanje cene poštnine nakupa - 1. del.

```
seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 500,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 500,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 499.99,
        VrstaDostave.PBS, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 300,
        VrstaDostave.PBS, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 499.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 300,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 499.99,
        VrstaDostave.PBS, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 300,
        VrstaDostave.PBS, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 499.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 300,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 499.99,
        VrstaDostave.PBS, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));
```

Slika 5.20: Izvorna koda testov za določanje cene poštnine nakupa - 2. del.

```
seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 300,
        VrstaDostave.PBS, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 499.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 300,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 499.99,
        VrstaDostave.PBS, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 300,
        VrstaDostave.PBS, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 499.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 300,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 299.99,
        VrstaDostave.PBS, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 0,
        VrstaDostave.PBS, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 299.99,
        VrstaDostave.HITRA_POSTA,VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(3));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(3));
```

Slika 5.21: Izvorna koda testov za določanje cene poštnine nakupa - 3. del.

```
seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 299.99,
        VrstaDostave.PBS, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 0,
        VrstaDostave.PBS, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 299.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 299.99,
        VrstaDostave.PBS, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 0,
        VrstaDostave.PBS, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 299.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(3));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(3));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 299.99,
        VrstaDostave.PBS, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 0,
        VrstaDostave.PBS, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 299.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));
```

Slika 5.22: Izvorna koda testov za določanje cene poštnine nakupa - 4. del.

```
seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.NAVADEN, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

// ------------------------------------------------------------

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 200,
        VrstaDostave.PBS, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 200,
        VrstaDostave.PBS, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 200,
        VrstaDostave.PBS, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 200,
        VrstaDostave.PBS, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 200,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 200,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 200,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 200,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 199.99,
        VrstaDostave.PBS, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 0,
        VrstaDostave.PBS, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));
```

Slika 5.23: Izvorna koda testov za določanje cene poštnine nakupa - 5. del.

```
seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 199.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(3));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(3));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 199.99,
        VrstaDostave.PBS, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 0,
        VrstaDostave.PBS, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 199.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 199.99,
        VrstaDostave.PBS, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 0,
        VrstaDostave.PBS, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 199.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(3));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(3));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 199.99,
        VrstaDostave.PBS, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));
```

Slika 5.24: Izvorna koda testov za določanje cene poštnine nakupa - 6. del.

```
seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 0,
        VrstaDostave.PBS, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(1));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 199.99,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.SILVER, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(2));

// ------------------------------------------------------

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.GOLD, 0,
        VrstaDostave.PBS, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.GOLD, 0,
        VrstaDostave.PBS, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.GOLD, 0,
        VrstaDostave.PBS, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.GOLD, 0,
        VrstaDostave.PBS, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.GOLD, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_OBROKIH));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.GOLD, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PO_PREVZETJU));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.GOLD, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREDRACUN));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));

seznamNakupov.add(ModelHelper.kreirajNakupRE(Status.GOLD, 0,
        VrstaDostave.HITRA_POSTA, VrstaPlacila.PREKO_SPLETA));
seznamPricakovanihCenPostnin.add(Double.valueOf(0));
```

Slika 5.25: Izvorna koda testov za določanje cene poštnine nakupa - 7. del.

```
    for (int i = 0; i < seznamNakupov.size(); i++) {

        NakupRE nakup = seznamNakupov.get(i);
        double pricakovanaCenaPostinte =
                seznamPricakovanihCenPostnin.get(i);

        try {

            NakupRE dejanskiNakup =
                    pp.pridobiCenoPostineNakupa(nakup, rulesClient);

            assertEquals(pricakovanaCenaPostinte,
                    dejanskiNakup.getCenaPostnine(), 0);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Slika 5.26: Izvorna koda testov za določanje cene poštnine nakupa - 8. del.

Tretja metoda *testDolocanjeDarilnegaBona* vsebuje primere za testiranje določanja vrednosti in vrste darilnega bona. Njena implementacija je prikazana na sliki 5.28. Na začetku testa definiramo dva seznama, v katera dodajamo objekte tipa *DarilniBonRE*. Objekti, ki jih dodajamo v prvi seznam, imajo nastavljeno vrednost atributa, ki določa skupen znesek nakupov. Objekti, ki jih dodajamo v drugi seznam, pa imajo nastavljeno vrednost atributov, ki določajo vrsto, ter vrednost darilnega bona in predstavljajo pričakovan rezultat ovrednotenja poslovnih pravil. Po kreiranju testnih primerov v zanki za vsak ustvarjeni bon iz prvega seznama pokličemo poslovna pravila ter pridobljen rezultat primerjamo s pričakovanim. Poslovna pravila pokličemo s pomočjo funkcije *pridobiDariniBon*, primerjavo vrednosti pa s klicem funkcije *assertEquals*.

```
@Test
public void testDolocanjeDarilnegaBona() {

    List<DarilniBonRE> darilniBoni =
            new ArrayList<DarilniBonRE>();
    List<DarilniBonRE> pricakovaniDarilniBoni =
            new ArrayList<DarilniBonRE>();

    darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(99.99));
    pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
            VrstaDarilnegaBona.ODSTOTKI, 0.0));

    darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(100));
    pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
            VrstaDarilnegaBona.ODSTOTKI, 0.05));

    darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(199.99));
    pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
            VrstaDarilnegaBona.ODSTOTKI, 0.05));

    darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(200));
    pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
            VrstaDarilnegaBona.ODSTOTKI, 0.10));

    darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(299.99));
    pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
            VrstaDarilnegaBona.ODSTOTKI, 0.10));

    darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(300));
    pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
            VrstaDarilnegaBona.ODSTOTKI, 0.15));

    darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(499.99));
    pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
            VrstaDarilnegaBona.ODSTOTKI, 0.15));
```

Slika 5.27: Izvorna koda testov za določanje darilnega bona - 1. del.

```
        darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(500));
        pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
                VrstaDarilnegaBona.ODSTOTKI, 0.20));

        darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(999.99));
        pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
                VrstaDarilnegaBona.ODSTOTKI, 0.20));

        darilniBoni.add(ModelHelper.kreirajDarilniBonREprazen(1000));
        pricakovaniDarilniBoni.add(ModelHelper.kreirajDarilniBonRE(
                VrstaDarilnegaBona.ODSTOTKI, 0.25));

        for (int i = 0; i < darilniBoni.size(); i++) {

            DarilniBonRE bon = darilniBoni.get(i);
            DarilniBonRE pricakovaniBon = pricakovaniDarilniBoni.get(i);

            try {
                DarilniBonRE dejanskiBon =
                        pp.pridobiDariniBon(bon, rulesClient);
                assertEquals(pricakovaniBon.getVrstaBona(),
                        dejanskiBon.getVrstaBona());
                assertEquals(pricakovaniBon.getVrednost(),
                        dejanskiBon.getVrednost(), 0);

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

Slika 5.28: Izvorna koda testov za določanje darilnega bona - 2. del.

Z zadnjo metodo *testOgledi Izdelka* testiramo poslovna pravila v zvezi s podražitvijo izdelka glede na njegove oglede v zadnji minuti. Implementacija metode je predstavljena na sliki 5.29. V njej najprej kreiramo objekt tipa *OgledIzdelkaEvent*, kateremu nastavimo vrednost atributa *idIzdelka* na 1. V vsakem prehodu zanke najprej pokličemo poslovna pravila (klic metode *zabeleziOgledIzdelka*) in tako zabeležimo ogled izdelka. V objektu, ki ga dobimo kot rezultat, je zapisano, za koliko naj se podraži izdelek glede na število ogledov v zadnji minuti. V pogojih if/else stavka so navedene omejitvene vrednosti, ki so definirane tudi v poslovnih pravilih. Kot v prejšnjih metodah smo tudi v tej pričakovano in dobljeno vrednost primerjali s pomočjo metode *assertEquals*. Ker smo v poslovnih pravilih definirali časovno drseče okno velikosti ene minute, se objekti, ki so bili v sejo dodani pred

več kot eno minuto, ne upoštevajo pri ovrednotenju pravil oz. se po eni minuti odstranijo iz seje in izbrišejo iz delavnega pomnilnika. Zato v nadaljevanju testa najprej ustavimo programsko nit za eno minuto in tako dosežemo izbris vseh objektov iz seje. Za tem zabeležimo še en ogled izdelka in preverimo, ali se podražitev nastavi nazaj na 0 odstotkov.

```
@Test
public void testOglediIzdelka() {

    OgledIzdelkaEvent ogledIzdelka = new OgledIzdelkaEvent(1);
    OgledIzdelkaEvent dejanskiOgledIzdelka = null;

    int stGeneriranihOgledov = 1100;
    long waitTime = 1000 * 60; // 1 min

    try {
        for (int i = 1; i <= stGeneriranihOgledov; i++) {
            dejanskiOgledIzdelka =
                    pp.zabeleziOgledIzdelka(ogledIzdelka, rulesClient);

            if (i < 400) {
                assertEquals(0, dejanskiOgledIzdelka.getPodrazitev(), 0);
            } else if (i < 1000) {
                assertEquals(0.02, dejanskiOgledIzdelka.getPodrazitev(), 0);
            } else if (i >= 1000) {
                assertEquals(0.05, dejanskiOgledIzdelka.getPodrazitev(), 0);
            }
        }

        try {
            Thread.sleep(waitTime);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        dejanskiOgledIzdelka =
                pp.zabeleziOgledIzdelka(ogledIzdelka, rulesClient);
        assertEquals(0, dejanskiOgledIzdelka.getPodrazitev(), 0);

    } catch (NapakaPriProzenjuPravil e) {
        e.printStackTrace();
    }
}
```

Slika 5.29: Izvorna koda testov za določanje podražitve izdelka glede na njegove oglede v zadnji minuti.

Z opisom testiranja smo zaključili poglavje o implementaciji poslovnih pravil. V naslednjem poglavju podamo zaključke naše diplomske naloge.

Poglavje 6

Zaključek

Organizacije gradijo poslovne informacijske sisteme, da z njimi podprejo in čim bolj avtomatizirajo poslovanje podjetja. Informacijski sistemi morajo biti zgrajeni tako, da delujejo v skladu z načeli in cilji podjetja. Ker se ti relativno hitro spreminjajo, mora biti na hitre spremembe prilagojen tudi podporni informacijski sistem. To je najlažje uresničljivo z implementacijo informacijskih sistemov na principu poslovnih pravil. Poslovna pravila predstavljajo jedro poslovnega informacijskega sistema. Zapisana morajo biti na pregleden in razumljiv način in upravljana tako, da se lahko čim hitreje prilagajajo spremembam v poslovanju podjetja, ne glede na hitrost razvoja ostale programske opreme.

Uvedba razvoja informacijskih sistemov na principu poslovnih pravil zahteva spremembe v vseh korakih razvoja programske opreme. Priporočljivo je, da se poslovna pravila smatrajo kot ločena entiteta razvoja, kar pomeni, da morajo imeti ločen življenjski cikel.

V diplomski nalogi smo preučili in predstavili sisteme BRMS in poslovna pravila ter opisali njuno uporabo in prednosti vpeljave na praktičnem primeru.

V drugem poglavju smo preučili in predstavili sisteme BRMS. Ugotovili smo, da so se razvili iz ekspertnih sistemov, ter da so prvi produkti temeljili na podatkovnih zbirkah. Opisali smo zgradbo modernega sistema BRMS, njihove prednosti in slabosti ter dejstva, ki se jih moramo zavedati in upoštevati pri njihovi uvedbi. Po našem mnenju je pri uvajanju sistema BRMS ključno izobraževanje vseh vključenih akterjev, saj bomo le tako lahko sistem BRMS uspešno vključili v informacijski sistem ter ga v polnosti izkoristili. Na koncu poglavja smo pred-

stavili še souporabo sistema BRMS z arhitekturnim konceptom SOA ter sistemom BPMS, saj smo mnenja, da njuna uporaba lahko bistveno pripomore k uspešni integraciji in izkoriščenosti sistema BRMS.

V tretjem poglavju smo predstavili poslovna pravila. Ugotovili smo, da se definicije razlikujejo glede na obdobje nastanka in avtorjev vidik, vsem pa je skupno to, da govorijo o določilih ali omejitvah poslovanja. Predstavili smo tudi najbolj pogoste načine zapisa poslovnih pravil, ki so se uveljavili zaradi intuitivnosti ter preglednosti glede na določeno vrsto sorodnih poslovnih pravil. Na koncu poglavja smo opisali tristopenjsko metodo ekstrakcije poslovnih pravil. Poudarili smo, da je za pridobitev popolnega nabora poslovnih pravil v velikem poslovnem sistemu potrebno poleg klasične poslovne analize izvesti tudi statično in dinamično ekstrakcijo pravil s pomočjo programskih orodij.

V četrtem poglavju smo predstavili dva komercialna ter dva odprtokodna produkta BRMS. Ugotovili smo, da so v zadnjih letih največji napredek doživeli odprtokodni produkti, ki so se razvili do te mere, da so primerni za velika poslovna okolja. Zapisali smo, da je največji dejavnik pri izbiri sistema BRMS možnost njihove integracije z informacijskim sistemom ter celotna cena njihovega upravljanja.

V zadnjem poglavju smo uporabo in prednosti vpeljave sistema BRMS predstavili na praktičnem primeru. Opisali smo postavitev sistema BRMS ter implementacijo poslovnih pravil na primeru spletne trgovine. Tekom implementacije poslovnih pravil smo ugotovili, da je mogoče nekatera poslovna pravila implementirati na več načinov. V našem primeru smo se pri implementaciji poslovnih pravil za določanje vrste in vrednosti darilnega bona odločili za implementacijo, ki omogoča hitrejše izvajanje. Druga možnost bi bila implementacija pravil, ki bi se izvajala počasneje, nudila pa bi veliko možnosti za razširitev v prihodnosti. Pri izbiri implementacije je najbolj pomembno to, da izbrana možnost najbolje ustreza podanim zahtevam.

Literatura

[1] Business rules group. defining business rules - what are they really? Dosegljivo: http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf, 2000. [Dostopano: 18. 7. 2016].

[2] The agile manifesto. Dosegljivo: http://dimsboiv.uqac.ca/8INF851/web/part1/introduction/The_Agile_Manifesto.pdf, 2001. [Dostopano: 28. 7. 2016].

[3] Manifesto for agile software development. Dosegljivo: http://agilemanifesto.org/, 2001. [Dostopano: 28. 7. 2016].

[4] Defect prevention: Reducing costs and enhancing quality. Dosegljivo: https://www.isixsigma.com/industries/software-it/defect-prevention-reducing-costs-and-enhancing-quality/, 2002. [Dostopano: 2. 8. 2016].

[5] Bpm, brms and soa delivering on the promise of organizational agility. Dosegljivo: http://www.slideshare.net/Zubin67/bpm-brms-and-soa?from_action=save, 2007. [Dostopano: 19. 7. 2016].

[6] Odm series 1: Ibm odm best practices – the abrd. Dosegljivo: http://blogs.perficient.com/ibm/2014/11/01/ibm-odm-best-practices/, 2014. [Dostopano: 19. 7. 2016].

[7] Business rule extraction in application modernization projects. Dosegljivo: http://www.donestes.com/business-rules/, 2015. [Dostopano: 1. 8. 2016].

[8] Answers to top brms questions. Dosegljivo: https://www.mercurymagazines.com/pdf/NCIBMBACORE1.pdf, 2016. [Dostopano: 18. 7. 2016].

[9] Apache maven project. Dosegljivo: https://maven.apache.org/, 2016. [Dostopano: 24. 7. 2016].

[10] Bm operational decision manager. Dosegljivo: http://www-03.ibm.com/software/products/en/odm, 2016. [Dostopano: 19. 8. 2016].

[11] Business process modeling. Dosegljivo: https://en.wikipedia.org/wiki/Business_process_modeling, 2016. [Dostopano: 19. 7. 2016].

[12] Domain-specific language. Dosegljivo: https://en.wikipedia.org/wiki/Domain-specific_language, 2016. [Dostopano: 19. 8. 2016].

[13] Drools documentation: Chapter 22. kie execution server. Dosegljivo: http://docs.jboss.org/drools/release/6.4.0.Final/drools-docs/html/ch22.html, 2016. [Dostopano: 23. 7. 2016].

[14] git. Dosegljivo: https://git-scm.com/, 2016. [Dostopano: 24. 7. 2016].

[15] Inrule technology. Dosegljivo: http://www.inrule.com/, 2016. [Dostopano: 19. 8. 2016].

[16] Inrule technology. Dosegljivo: https://en.wikipedia.org/wiki/InRule_Technology, 2016. [Dostopano: 19. 8. 2016].

[17] Installing and configuring an ibm operational decision management golden topology. Dosegljivo: http://www.ibm.com/developerworks/bpm/bpmjournal/1305_defreitas/1305_defreitas.html#resources, 2016. [Dostopano: 20. 8. 2016].

[18] jbpmhomepage. Dosegljivo: http://www.jbpm.org/, 2016. [Dostopano: 21. 7. 2016].

[19] Knowledge management. Dosegljivo: https://en.wikipedia.org/wiki/Knowledge_management, 2016. [Dostopano: 18. 7. 2016].

[20] Mycin. Dosegljivo: `https://en.wikipedia.org/wiki/Mycin`, 2016. [Dostopano: 16. 7. 2016].

[21] Openrules - business rules and decision management system. Dosegljivo: `http://openrules.com/`, 2016. [Dostopano: 19. 8. 2016].

[22] Optaplanner. Dosegljivo: `http://www.optaplanner.org/`, 2016. [Dostopano: 21. 7. 2016].

[23] Service-oriented architecture. Dosegljivo: `https://en.wikipedia.org/wiki/Service-oriented_architecture`, 2016. [Dostopano: 19. 7. 2016].

[24] Daniel S. Appleton. Business rules: The missing link. *Datamation*, (10):145–150, 1984.

[25] Michal Bali. *Drools JBoss Rules 5.0 Developer's Guide*. 2009, publisher=Packt Publishing Ltd.

[26] Jerome Boyer and Hafedh Mili. *Agile Business Rule Development*. Springer, 2011.

[27] William J. Clancey. The epistemology of a rule-based expert system —a framework for explanation. *Artificial Intelligence*, 20(3):215 – 251, 1983.

[28] Ian Graham. *Business Rule Management and Service Oriented Architecture*. John Wiley & Sons, Ltd, 2006.

[29] Ian Graham. *Business Rules Management and Service Oriented Architecture: A Pattern Language*. John Wiley & Sons, Ltd, 2006.

[30] P. E. Hart, R. O. Duda, and M. T. Einaudi. Prospector—a computer-based consultation system for mineral exploration. *Journal of the International Association for Mathematical Geology*, 10(5):589–610, 1978.

[31] Robert K. Lindsay, Bruce G. Buchanan, Edward A. Feigenbaum, and Joshua Lederberg. Dendral: A case study of the first expert system for scientific hypothesis formation. *Artificial Intelligence*, 61(2):209 – 261, 1993.

[32] Tony Morgan. *Business Rules and Information Systems: Aligning IT with Business Goals*. Adison-Weasly, Inc, 2002.

[33] Ronald G. Ross. *Entity Modeling: Techniques and Application.* Business Rule Solutions Inc, 1987.

[34] Ronald G. Ross. *The Business Rule Book (2nd edition).* Business Rule Solutions, LCC, 1997.

[35] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6):26–36, 2000.

[36] Barbara von Halle. *Business Rules Applied: Building Better Systems Using the Business Rules Approach.* John Wiley & Sons, Inc., 2001.

[37] Chen Zhang, Thomas O Meservy, E Ted Lee, and Jasbir Dhaliwal. An exploratory case study of the benefits of business rules management systems. *ICIS 2009 Proceedings*, page 19, 2009.

[38] Michael zur Muehlen, Marta Indulska, and Gerrit Kamp. Business process and business rule modeling languages for compliance management: A representational analysis. In *Tutorials, Posters, Panels and Industrial Contributions at the 26th International Conference on Conceptual Modeling - Volume 83*, ER '07, pages 127–132, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.